MSX-Datapack

エムエスエックス・データパック turbo R 版

Volume 3

ASCII

はじめに

このたびは、「MSX Datapack turbo R 版」をお買い上げいただきまして、誠にありがとうございます。「MSX-Datapack turbo R 版」は MSX turbo R の仕様と解説したマニュアルと、サンプルプログラムとをセットにしたパッケージです。すでに発売している「MSX-Datapack」は、MSX、MSX_2、MSX_{2+}の仕様を解説したもので、「MSX Datapack turbo R 版」はその続編という位置づけになっています。したがって、このパッケージで解説している内容は、MSX turbo R で追加・変更された仕様です。従来と互換性のある部分(BASIC や BIOS、VDP など)については、「MSX-Datapack」をご参照下さい。

さて、1983 年に登場した MSX ホームパーソナルコンピュータは、MSX_2、MSX_{2+}へと主にグラフィックを強化して来ました。そして、MSX turbo R では、Z80 互換の新 CPU R800 を搭載することにより、平均 10 倍の高速処理を実現しつつ、従来の MSX との上位互換性を保っています。

MSX の互換性を守ってゆくために、ハードウェアおよびソフトウェアを作成するときはこのパッケージの内容をルールとして守って下さい。

このパッケージには以下のものが含まれています。

MSX-Datapack turbo R 版マニュアル	1 冊
MSX-Datapack turbo R 版サンプルディスク (3.5-2DD フロッピーディスク)	1 枚

ご注意

1. このソフトウェアおよびマニュアルの一部または全部を株式会社アスキーの文書による許可なくして複製することは、メディアの形態を問わず禁じます。
2. このソフトウェアおよびマニュアルは個人として利用するほかは、著作権上、株式会社アスキーに無断で使用することはできません。
3. このマニュアルに記載されている事柄は、将来予告なく変更することがありますが、当社に登録されている方にはご案内をお送り致します。
4. 製品の内容については万全を期しておりますが、製品の内容についてのご不審や誤りマニュアルの記載もれなど、お気づきのことがございましたら、マニュアルの巻末の「お問い合わせについて」の要領で問い合わせ下さい。
5. このソフトウェアを運用した結果の影響については、4項にかかわらず、責任を負いかねますのでご了承下さい。

このマニュアルの表記法

1. [A] などはキーボードに刻印されているとおりに表記します。
2. リターンキーは [RETURN] または [↵]、ファンクションキーは [F1] などと表記します。
3. 「... を押しながら ... を押す」といったキー操作に関しては

   [SHIFT] + [F1]

   [CTRL] + [STOP]

   のように表記します。
4. ディスプレイに表示される文字に関しては

   ```
   ^A
   ```

   のように囲みます。この例では、画面に [CTRL] + [A] が表示されたことを示します。
5. MSX turbo R では、MSX_{2+}などと同じく、BIOS が MAIN ROM と SUB ROM に分割されています。これを区別するため、以下のように表記します。

   CHGET(009FH/MAIN)　MAIN ROM の 009FH 番地
   REDCLK(01F5H/SUB)　SUB ROM の 01F5H 番地
6. ワークエリアおよびフックは、以下のように表記します。

   【VALTYP(0F663H.1)】　0F663H 番地の 1 バイトを使用
   【BUF(0F55EH,258)】　0F55EH 番地以降の 258 バイトを使用
7. 原則的に入力する文字は大文字、小文字を問いませんが、大文字と小文字の区別が必要なときは、マニュアルにその旨を記載します。全角文字と半角文字は必ず区別します。
8. 知っていると便利な情報は

   のように囲みます。

9. このマニュアル中で示される MSX の画面は、レイアウトなどの都合上、縦横比が変わっていることがあります。あらかじめご了承下さい。

10. その他は必要に応じて解説します。

目次

第 1 部　MSX turbo R

第2部 MSX-DOS2

第3部 MSXView

第 4 部 MSX-MIDI

APPENDIX

第1部 MSX turbo R

1章
MSX turbo Rとは

MSX turbo R は、MSX_{2+}の上位互換機種で、以下のような特徴があります。

1. 新CPU
   CPUには、従来のZ80に加え、新しいCPU（R800）を採用し、5～20倍（平均10倍、対MSX_{2+}比）の高速処理を実現しました。

2 MSX-DOS2
   MSX-DOS1と共に、すでに定評のあるMSX-DOS2を標準で搭載しました。これにより、

   - MS-DOS 2.11と互換性のあるツリーディレクトリのサポート
   - 大容量のRAMの統括的な管理（RAMディスクなど）
   - 大容量ディスクの管理（ハードディスクなど）
   - 環境変数によるカスタマイズ

   などが可能になりました。

3. 256KBメインRAM以上
   メインRAM容量は256Kbyte以上を標準実装としました。このRAMは、RAMディスク プログラムエリア、およびアプリケーションのデータエリアとして使用できます。

4. PCM
   PCMの録音・再生機能を標準搭載しました。これにより、従来のFM音源による音楽だけでなく、人の声やリアルな効果音の録音・再生が可能になりました。

2章 システム構成

この章では、新 CPU の R800 を始めとした、MSX turbo R のハードウェア構成を説明します。

2.1 基本仕様

MSX turbo R と MSX_{2+} との基本的な仕様の違いを一覧表で示します。

表 1.1 MSX turbo R と MSX_{2+} との概略仕様一覧

		MSX turbo R	MSX_{2+}
CPU		R800 相当品 (7.15909MHz)	Z80A 相当品 (3.579545MHz)
メモリ	ROM	160KB (MSX BASIC ver.4) MAIN ROM 32KB SUB ROM 16KB 漢字ドライバ 32KB MSX-DOS1 16KB MSX-DOS2 48KB MSX-MUSIC 16KB	96KB (MSX BASIC ver.3) MAIN ROM 32KB SUB ROM 16KB 漢字ドライバ 32KB MSX-DOS1 16KB (オプション) (オプション)
	RAM	256KB 以上	64KB 以上
	VRAM	128KB	

		MSX turbo R	MSX_{2+}
画面表示	VDP	V9958相当品	
	解像度（最大）	512×212（ノンインタレース時）	
		512×424（インタレース時）	
	表示色（最大）	19268色	
	ハードウェア スクロール	縦・横	
カセットインターフェイス		なし	FSK方式 1200・2400bps
サウンド	PSG	AY-3-8910相当品	
	FM音源	MSX-AUDIO（オプション）	
		MSX-MUSIC	（オプション）
	MIDI	MSX-MIDI（オプション）	なし
キーボード		英数、ひらがな、カタカナ、グラフィック文字対応	
		JIS配列・50音配列対応	
フロッピーディスク		3.5インチ2DD	
		（1DDは読み書き可）	
	フォーマット	MS-DOS 2.11準拠	
プリンタ		8ビットパラレル	
		セントロニクスインターフェイス準拠	
カートリッジスロット		カートリッジバスを1つ以上持つ	
		ROMカートリッジ、拡張バスに対応するスロット	
ジョイスティック		2	
漢字機能	漢字ROM	第1水準	
		第2水準（オプション）	
	漢字入力	単漢変換（MSX-JE対応）	
リアルタイムクロック		RP5C01相当品	

2.2 ハードウェア構成

MSX turbo R のハードウェア構成について説明します。

2.2.1 主要LSI

MSX turbo R は、主に以下の LSI を使用します。

CPU	R800 相当品（クロック 7.15909MHz）
	Z80A 相当品（クロック 3.579545MHz）
VDP	V9958 相当品
PSG	AY-3-8910 相当品
FM 音源	MSX MUSIC（YM2413 相当品）
MIDI	MSX-MIDI（i8251 相当品と、i8253 または i8254 相当品）
PPI	i8255 相当品
システム IC	S1990 相当品

2.2.2 メモリ

MSX turbo R は以下のメモリから構成されます。

ROM	MSX BASIC ver.4		80KB
	(MAIN ROM	32KB)	
	(SUB ROM	16KB)	
	(漢字ドライバ	32KB)	
	MSX-DOS1		16KB
	MSX-DOS2		48KB
	MSX-MUSIC		16KB
RAM	メイン RAM		256KB 以上（メモリマッパー使用）
	VRAM		128KB

2.2.3 ブロック図

MSX turbo R のシステム構成をブロック図で説明します。

MSX turbo R では 2 つの CPU（Z80・R800）を搭載しました。この 2 つの CPU はプログラムの実行中に、自由に切り換えることができます。片方の CPU が動作しているときはもう片方は HOLD 状態になります。2 つの CPU が同時に動作することはありません。

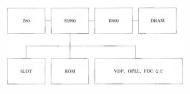



図 1.1 MSX turbo R のブロック図

S1990 は、CPU の切り換えを実現するためのシステムコントロール LSI です。S1990 は その他にメモリやスロットの制御、ウエイト信号の発生、および I/O デコードなどのコントロールを行います。

2.3 スロット構成

MSX turbo R では、スロット構成が標準化されました。これは、メイン RAM が R800 に直接つながっているか、それともスロットに接続されているかによって、システム性能が大きく変わるためです。また、アプリケーションソフトウェアの開発を容易にする、という目的もあります。

DOS は MSX-DOS1 と MSX-DOS2 との両方を搭載しました。スロット 3-2 のページ 1 に 4 枚のローカルなバンクをもって、前の 3 ページが MSX-DOS2、後ろの 1 ページが MSX-DOS1 となっています。

システム的には、2 つの CPU と 2 つの DOS との組合せは自由ですが、システムを単純にして、ユーザーの混乱を避けるため、標準的には

- Z80 と DOS1 (Z80 モード)
- R800 と DOS2 (R800 モード)

という 2 つの組合せをサポートします。

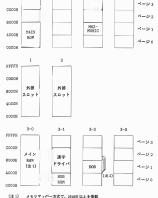
このモードの切り換えはシステム起動時に、挿入されているカートリッジやディスケットによって、自動的に判断して行われます。

（注1）　メモリマッパー方式で、256KB以上を搭載
（注2）　ローカルバンク。前の3ページがMSX-DOS2、後ろの1ページがMSX-DOS1

図 1.2 MSX turbo Rのスロット構成

2.4 R800

2.4.1 R800の特徴

R800はMSX turbo R用に開発されたCPUで、以下のような特徴があります。

1. Z80とオブジェクトコンパチブル
   Z80用に書かれたソフトウェアは、CPUのタイミングに依存する部分を除いて、変更なしに動作します。

2. CPUクロック7.15909MHz
   命令あたりのクロック数がZ80に比べて大幅に減少しているため、Z80換算では28.63636MHzに相当します（ノーウエイト時）。

3. 乗算命令
   8bit × 8bit → 16bit および 16bit × 16bit → 32bit の精度を持つ乗算命令をサポートしました。[illegible]演算処理速度の[illegible]が可能です。

4. 未定義命令
   Z80では未定義だったIX/IYレジスタの上位・下位8ビットごとのアクセスを正式に保証しました。

表1.2で、主なインストラクションの動作スピードを示します（MSX turbo Rはノーウエイト時。MSX_{2+}はM1サイクルに1ウエイト入っていることを考慮）。

2.5 ウエイト

MSX turbo Rでは、R800は常にノーウエイトで動作しているわけではありません。以下の場合には、ウエイトサイクルが挿入されます。

1. 外部スロットをアクセスするとき（3ウエイト）
   外部スロットへアクセスするときには、今までの周辺機器をサポートするために、MSX_{2+}と同じアクセスタイミングでアクセスするように設計されています。

2. 内部ROMをアクセスするとき（2ウエイト）
   内部ROMとは、BIOS、BASIC、および内蔵ディスクROMなどのシステムソフトウェアを格納しているROMのことです。ただし、MSX turbo Rは、初期状態ではBIOS、BASIC、SUB ROM、および漢字ドライバの前半の16KB（合計64KB）をメインRAMにコピーして動作するので、ほとんどの場合は、次に述べる3.がウエイトの条件になります。

表 1.2 MSX turbo R と MSX_{2+} との動作比較

命令		MSX_{2+} (単位μs)	MSX turbo R (単位μs)	倍率
LD	r,s	1.40	0.14	× 10.0
LD	r,(HL)	2.23	0.42	× 5.3
LD	r,(IX+n)	5.87	0.70	× 8.4
PUSH	qq	3.35	0.56	× 6.0
LDIR	(BC <> 0)	6.43	0.98	× 6.6
ADD	A r	1.40	0.14	× 10.0
INC	r	1.40	0.14	× 10.0
ADD	HL,ss	3.35	0.14	× 24.0
INC	ss	1.96	0.14	× 14.0
JP		3.07	0.42	× 7.3
JR		3.63	0.42	× 8.7
DJNZ	(B <> 0)	3.91	0.42	× 9.3
CALL		5.03	0.84	× 6.0
RET		3.07	0.56	× 5.5
MULTU	A,r	160	1.96	× 82
MULTUW	HL,rr	361	5.03	× 71

3. 内部 DRAM がページブレークを起こしたとき（1 ウエイト）
   R800 は、「ページアクセスモード」をサポートする、DRAM 専用のバスを持っています。そのため、DRAM の性能をフルに引き出すことができます。
   ページアクセスモードとは、アドレスの下位 8 ビットだけが変化するような連続したメモリアクセスに対して、高速にアクセスすることができるモードです。下位 8 ビット以外が変化することを「ページブレークを起こした」と表現します。
   ページブレークを起こすのは、平均すると DRAM への 2 回のアクセスに 1 回くらいなので、実質的に約 0.5 ウエイトで動作することになります。
   ページブレークは DRAM リフレッシュ時にも発生します。R800 では Z80 と違って DRAM リフレッシュが命令の実行とは非同期に行われます。MSX turbo R では、リフレッシュは 31μs 毎に実行され、1 回のリフレッシュに 280ns かかります。また、ジャンプ命令（相対ジャンプ、絶対ジャンプとも）では、必ずページブレークが発生します。

そこで、高速化が必要なプログラムでは、以下のような点を注意すればよいことになります。

1. プログラムは RAM に転送する
   ROM カートリッジのプログラムは、高速化が必要な部分を RAM に転送して実行すると、高速に動作します。フロッピーディスクで供給されるプログラムは、必然的に RAM 上で動作するので問題ありません。

2. ページブレークを起こさないようにコーディングする
   RAM 上で動作させる場合、高速化が必要なループ中では、なるべく CPU のレジスタ

だけですませて、データメモリへのアクセス（スタックも含む）が少なくなるようにコーディングすると、ページアクセスモードの利点を最大限に生かすことができます。また、ループが xx00H〜xxFFH までにおさまるようにプログラミングするといっそう効果的です。

3 システムタイマーを使う

MSX turbo R では、前述のようにページブレークやリフレッシュサイクルなどの影響で、MSX_{2+}のように「コーディング段階で命令の実行時間が正確に判る」ことがありません。したがって、MSX turbo R でも、MSX_{2+}でも動作するようなプログラムを作成するためには、ソフトウェアループでタイミングをとることは勧められません。MSX turbo R では、3.91μs でカウントアップするシステムタイマーを搭載しているので、これを利用してタイミングをとって下さい。

3章 システムの動作モード

3.1 CPUのモード

MSX turbo Rには、MSX-DOS2とMSX-DOS1との両方が内蔵されています。自動的に起動するアプリケーションプログラムは、どちらの環境で動作するかを選択することができます。また、起動するDOSまたはDisk BASICのバージョンによって、システムがR800モードになるか、Z80モードになるかが異なります。その組み合わせは、以下のとおりです。

表 1.3 システムのバージョンとCPUモードとの組み合わせ

バージョン	モード
MSX-DOS2またはDisk BASIC 2.0	R800 DRAMモード
MSX-DOS1またはDisk BASIC 1.0	Z80モード

この切り換えは、システムソフトウェアが自動的に行います。これらは、アプリケーションソフトウェアがBIOSの【CHGCPU(0180H/MAIN)】をコールして、CPUを切り換えることもできます。

以下では、CPUの動作モードについて説明します。

3.1.1 R800 ROMモードとR800 DRAMモード

MSX turbo Rに搭載されているCPUのR800は、DRAM上のプログラムやデータをROMより速く読み書きできる機能を持っています。したがって、ROM上にあるBIOSやBASICなどのシステムソフトウェアをDRAM上に置いて実行すれば、動作速度が速くなります。そのため、MSX turbo Rには、システムソフトウェアをDRAMに置いて実行する機能があります。

システムソフトウェアが、ROMから読み出されて実行されている状態を「R800 ROMモード」、DRAMから読み出されて実行されている状態を「R800 DRAMモード」と呼びます。

MSX turbo R では DRAM上におかれたシステムソフトウェアは、ROM上におかれたシステムプログラムと同じスロットに見えるハードウェアになっている（S1990が実現している）ので、アプリケーションプログラム側では、R800ROMモードかR800 DRAMかを意識する必要はありません。

R800 DRAMモードのときは、システムソフトウェアのあるDRAMはライトプロテクトされているので、RAMとしてアクセスするときはできません。システムソフトウェアとして読めるだけです。

MSX turbo RはDRAMをメモリマッパーをとおしてアクセスしています。システムソフトウェアのあるDRAMは、メモリマッパー上に実装されているDRAMの最後の4セグメントです。例えば、MSX turbo R本体に内蔵されているRAMが256KBならば、全セグメント数は256KB÷16KB=16セグメントとなり、セグメント番号12～15のセグメントが使われることになります。

また この4セグメントは本体内蔵のDRAMのサイズ毎にイメージが見えます。256KBであれば、セグメント番号12～15のセグメントがセグメント番号28～31、44～47、60～63にも見えます。内蔵RAMが512KBであれば セグメント番号28～31のセグメントがセグメント番号60～63にも見えます。

つまり、システムプログラムの転送先は、本体に内蔵されたRAM容量にかかわらず、セグメント番号60～63になるわけです。

Z80モードやR800 ROMモードのときは、この4つのセグメントは未使用になります。この領域の使い方は、6章「マッパーRAMセグメント」を参照して下さい。

3.1.2 Z80モード

Z80モードでは、Z80 CPUが動作します。したがって、システムの動作速度はMSX_{2+}と同じになります。

3.2 ソフトウェアの起動

アプリケーションプログラムには、大きく分けて

1. ROMカートリッジの機械語プログラム
2. ROMカートリッジのBASICプログラム
3. フロッピーディスクの機械語プログラム
4. フロッピーディスクのBASICプログラム

の4種類があります。以下では、この4種類のプログラムについて、どの様に日本語MSX DOS2とMSX DOS1とが選択されるかを説明します。

3.2.1 ROM カートリッジの機械語プログラム

Disk BASIC 1.0 で、ROM カートリッジの機械語プログラムを起動するためには、いままでと同じように、カートリッジの INIT ルーチンで、起動させたいプログラムへのインタースロットコールを【H.STKE(0FEDAH)】に書いて、リターンします。[illegible]、すべてのカートリッジの初期設定が終わり、[illegible]が起動す[illegible] BASIC 1.0 の環境になります。

Disk BASIC 2.0 でアプリケーションプログラムを起動するためには、カートリッジの INIT ルーチンが、【H.STKE(0FEDAH)】を設定するときに、【USRTAB(0F39AH)】に 074H を【USRTAB+1】に 064H を書き込み、リターンします。すると、すべてのカートリッジの初期設定が終わり、アプリケーションプログラムが起動するときは、Disk BASIC 2.0 の環境になります。

ROM カートリッジの機械語ソフトウェアを起動させる手順の詳細は、Volume 1 の 第 2 部 7.10.3「オートスタート」と Volume 2 の 第 5 部 3 章「カートリッジソフトの作成法」とか、「MSX_2 テクニカルハンドブック」の P.87 と P.328~P.339 とを参照して下さい。なおカートリッジソフトの作成についての記述は 『MSX Datapack』 と 『MSX_2 テクニカルハンドブック』とは同じです。

3.2.2 ROM カートリッジの BASIC プログラム

Disk BASIC 1.0 で、ROM カートリッジの BASIC プログラムを起動するためには、今までと同じように、ヘッダーの TEXT に BASIC プログラムの番地を設定します。

Disk BASIC 2.0 で BASIC プログラムを起動するためには、それに加えて、ヘッダーの INIT ルーチンで、【USRTAB(0F39AH)】に 074H を、【USRTAB+1】に 064H を書き込みます。

ROM カートリッジの機械語ソフトウェアを起動させる手順の詳細は、Volume 1 の 第 2 部 7.10.3「オートスタート」と Volume 2 の 第 5 部 3 章「カートリッジソフトの作成法」とか、『MSX_2 テクニカルハンドブック』の P.87 と P.328~P.339 とを参照して下さい。なおカートリッジソフトの作成についての記述は、『MSX-Datapack』と『MSX_2 テクニカルハンドブック』とは同じです。

3.2.3 フロッピーディスクの機械語プログラム

AUTOEXEC.BAT による立ち上げ

MSX-DOS1 で、AUTOEXEC.BAT によりプログラムを起動するためには、MSX-DOS1 でフォーマットしたフロッピーディスクに、MSXDOS.SYS と COMMAND.COM とをコピー[illegible] AUTOEXEC.BAT およびアプリケーションプログラムを作成(またはコピー)します。

MSX-DOS2 で、AUTOEXEC.BAT によりプログラムを起動するためには、MSX-DOS2 でフォーマットしたフロッピーディスクに、MSXDOS2.SYS と COMMAND2.COM とをコ

ピーし、AUTOEXEC.BAT およびアプリケーションプログラムを作成（またはコピー）します。

ブートセクタによる立ち上げ

MSX-DOS1で、ブートセクタによりプログラムを起動するためには、MSX-DOS1でフォーマットしたフロッピーディスクのブートセクタの001EH～00FFHに、アプリケーションプログラムを読み込んで実行させるプログラムを書き込みます。

最初に、001EH～00FFHに書き込まれたプログラムは、0C01EH～0C0FFHに読み込まれ、キャリーフラグを0にしてコールされます。この時点では、ページ1をRAMに切り換えることができないので、ユーザープログラムの起動には不適当です。次に、もう1度MSX-DOS1の環境下で、キャリーフラグを1にしてコールされます。このとき、スロットの状態は以下のようになっています。

ページ	内容
0	RAM
1	Disk ROM
2	RAM
3	RAM

また、レジスタには以下の情報が入っています。

レジスタ	内容
A	0ならば電源投入直後を示す。
DE	この内容をコールすると、ページ1のDisk ROMがRAMに切り換わる。
HL	ディスクエラー処理ルーチンへのポインタへのポインタ（MSX-Datapack Volume 1の第2部7.9.3「エラー処理」参照）。

MSX-DOS2で、ブートセクタによりプログラムを起動するためには、MSX-DOS2でフォーマットしたフロッピーディスクのブートセクタの0030H～00FFHに、アプリケーションプログラムを読み込んで実行させるプログラムを書き込みます。

最初に、0030H～00FFHに書き込まれたプログラムは、0C030H～0C0FFHに読み込まれキャリーフラグを0にしてコールされます。この時点では、ページ1をRAMに切り換えることができないので、ユーザープログラムの起動には不適当です。次に、もう1度MSX-DOS2の環境下で、キャリーフラグを1にしてコールされます。そのときのスロットの状態やレジスタの情報は、MSX-DOS1と同じです。

MSX-DOSのブートシーケンスについての詳細は、MSX-Datapack Volume 1の第3部3.1「MSX-DOSの起動」か、『MSX_2 テクニカルハンドブック』のP.97～98を参照して下さい。

3.2.4 フロッピーディスクの BASIC プログラム

Disk BASIC 1.0 で、BASIC プログラムを起動するためには、MSX-DOS1 でフォーマットしたディスクに、AUTOEXEC.BAS およびアプリケーションプログラムを作成（またはコピー）します。MSXDOS.SYS と COMMAND.COM とは入れてはいけません。

Disk BASIC 2.0 で、BASIC プログラムを起動するためには、MSX-DOS2 でフォーマットしたディスクに、AUTOEXEC.BAS およびアプリケーションプログラムを作成（またはコピー）します。MSXDOS2.SYS と COMMAND2.COM とは入れてはいけません。

4章
BASIC

4.1 追加された命令

MSX turbo R では、PCM機能などが追加されたので、BASIC でもそれらの機能が使えるよう、CALL PAUSE、CALL PCMPLAY、CALL PCMREC 命令が追加されました。
以下では、それぞれの命令について説明します。

CALL PAUSE

機能　指定された時間だけポーズします。

書式　CALL PAUSE(ポーズ時間)

文例　CALL PAUSE(1000)　　1秒間ポーズ

解説　指定された時間だけ、BASICプログラムの実行を停止します。ポーズ時間の単位はミリ秒です。CALL PAUSE 実行中は、割り込みは許可されています。CALL PAUSE 実行中でも、[CTRL] + [STOP] キーでプログラムを中断することができます。CPU の実行速度の影響を受けないタイミングをとるときに使います。

CALL PCMPLAY

機能　PCM音を再生します。

書式　CALL PCMPLAY(@開始番地, 終了番地, サンプリングレート)
メイン RAM からの再生
CALL PCMPLAY(@開始番地, 終了番地, サンプリングレート, S)
VRAM からの再生
CALL PCMPLAY(配列変数名, [長さ], サンプリングレート)
配列変数からの再生

文 例

CALL PCMPLAY(@&HB000, &HBFFF, 1)
メインメモリの&HB000番地〜&HBFFF番地をPCMデータとして再生します。サンプリングレートは7.875KHzとします。

解 説

メインRAMまたはVRAMの内容をPCMデータとして、指定されたサンプリング周波数で再生します。

- サンプリングレート
  サンプリングレートを指定します。

値	サンプリングレート (KHz)
0	15.75
1	7.875
2	5.25
3	3.9375

- 長さ
  省略できます。省略したときは、配列変数の内容すべてを再生します。

データの形式はアブソリュートバイナリで、1〜255が通常のデータです。0は特殊なデータで、0の後に続く1バイトで指定された回数分、0レベル(127)を出力します。

注 意

Z80モードのときには、自動的にR800モードに切り換えて実行し、終わるとZ80モードに戻ります。再生中に、[STOP]キーが押されるとプログラムの実行を中断し、BASICに戻ります。
MSX-BASICでは、&Hで表現できる16進数は&H0000〜&HFFFFの4桁であることと、&H8000〜&HFFFFは負の数であることとに注意して下さい。これは、メインRAMを指定する際には問題にはなりませんがVRAMを指定するときには注意が必要です。例えば、VRAMの後半の64KBを再生するには

```
CALL PCMPLAY(@65536,131071,2,S)
```

のように指定しなければなりません(131071=65536×2-1)。これを

```
CALL PCMPLAY(@&H10000,&H1FFFF,2,S)
```

と指定すると、「Overflow」エラーになります。また

```
CALL PCMPLAY(@&HFFFF+1,&H1FFFF×2+1,2,S)
```

は、&HFFFF+1=-1+1=0と&HFFFF×2+1=-1×2+1=-1=&HFFFFとで、CALL PCMPLAY(@0,&HFFFF,2,S)と同じになります。

CALL PCMREC

機能	音声をPCM録音します。

書式

CALL PCMREC(@開始番地, 終了番地, サンプリングレート [,[トリガレベル][, 圧縮スイッチ]]) **メインRAMへの録音**
CALL PCMREC(@開始番地,終了番地, サンプリングレート ,[トリガレベル],[圧縮スイッチ], S) **VRAMへの録音**
CALL PCMREC(配列変数名, [長さ], サンプリングレート [,[トリガレベル][, 圧縮スイッチ]]) **配列変数への録音**

文例

```
CLEAR 300, &HB000
CALL PCMREC(@&HB000. &HBFFF. 1)
```

メインメモリの&HB000番地~&HBFFF番地に音声をPCM録音します。サンプリングレートは7.875KHzとします。

解説

音声をPCM方式で、指定された周波数でサンプリングし、メインRAM、VRAM、または配列変数に録音します。

- サンプリングレート
  サンプリングレートを指定します。

値	サンプリングレート (KHz)
0	15.75
1	7.875
2	5.25
3	3.9375

- トリガレベル
  録音開始時の入力レベルを設定します。値の範囲は、0~127です。省略されたときや0のときは、ただちに録音を開始します。

- 圧縮スイッチ
  無音データ圧縮を行うかどうかを指定します。

値	意味
省略	圧縮しない
0	圧縮しない
1	圧縮する

- 長さ
  省略できます。省略したときは、配列変数の内容すべてに録音します。

データの形式はアブソリュートバイナリで、1～255が通常のデータです。0レベル付近（126～128）のデータが2つ以上連続したときは、0とその連続した回数を記録することによって、データを圧縮できます。この圧縮を無音データ圧縮といいます。

注　意

Z80モードのときには、自動的にR800モードに切り換えて実行し、終わるとZ80モードに戻ります。また、Z80モードやR800 ROMモードのときに、サンプリングレートを15.75KHzに指定すると、「Illegal function call」エラーになります。録音中に、STOPキーが押されるとプログラムの実行を中断し、BASICに戻ります。
VRAMへ録音するときは、CALL PCMPLAYと同じように、指定する番地に注意して下さい。

4.2　変更された命令

MSX turbo Rでは、MSX-DOS2が標準搭載され、カセットインターフェイスが削除されたので、ファイル名をパラメータとする命令の仕様が変わりました。
以下では、それぞれの命令について説明します。

BLOAD/BSAVE/LOAD/MERGE/OPEN/RUN/SAVE

解　説

これらの命令で、ファイル名に「CAS:」を指定すると、「Bad file name」エラーになります。また、MSX DOS2で、ディスクファイルを指定するときは、ファイル名をフルパスで指定することができます。

COPY/FILES/KILL/LFILES/NAME

解　説

MSX-DOS2に対応しました。MSX-DOS2を使っているときは、ファイル名をフルパスで指定することができます。

4.3 削除された命令

MSXturbo R では、カセットインターフェイスが削除されました。それにともなって、カセットテープ関連の命令が削除されました。
以下では、それぞれの命令について説明します。

CLOAD/CSAVE/MOTOR

解　説	これらの命令を実行すると、「Syntaxerror」になります。

5章
BIOS

5.1 追加されたエントリ

MSX turbo R では、BIOS についても、CPU の切り換えや PCM 関連のエントリなどが追加されています。

以下では、それぞれのエントリについて説明します。

5.1.1 CPU 切り換え

CPUの切り換えの BIOS には、CHGCPU と GETCPU というエントリがあります。

CHGCPU(0180H/MAIN)

機　能	CPU を切り換えます。

コール手順	A

- モード

CPUのモードを指定します。

モード	意味
00	Z80モード
01	R800 ROMモード
10	R800DRAMモード
11	システム予約

- LED

LEDの状態を指定します。

LED	意味
0	LEDを変化させない
1	LEDを変化させる

戻り値　なし

変更レジスタ　なし

解説　Aレジスタの内容によって、CPUを切り換えます。もし、Aレジスタのビット7が1なら、変更したCPUにあわせて、CPUの状態を表すLEDを変化(点灯するか消灯するか)させます。Aレジスタのビット7が0なら、LEDを変化させません。
変更前のCPUのレジスタの内容は、Rレジスタ以外は、変更後のCPUに受け継がれます。変更後は、割り込みは許可されます。

注意　CPUの切り換えは、システムコントロールLSI (S1990) 内の設定を変えるだけです。そのため、Z80やR800 ROMモードに切り換えた後、空いたDRAMエリアの内容を書き換えて、R800 DRAMモードにするとシステムは暴走します。DRAMへのシステムソフトウェアの転送は、システム起動時にのみ行っています。

GETCPU(0183H/MAIN)

機　能　　現在、どちらのCPUが動作しているかを調べます

コール手順　　なし

戻り値　　A

値	意味
0	Z80 モード
1	R800 ROM モード
2	R800 DRAM モード

変更レジスタ　　F

5.1.2 PCM

PCM データの録音・再生の BIOS には、PCMPLY と PCMREC というエントリがあります。

PCMPLY(0186H/MAIN)

機　能　　PCM のデータを再生します。

コール手順　　A

b7	b6	b5	b4	b3	b2	b1	b0
R	0	0	0	0	0	S	S

b1, b0 (S S): サンプリングレート
b6〜b2: 常に0
b7 (R): VRAM/メインメモリ

- サンプリングレート

サンプリングレートを指定します。

サンプリングレート	意味 (KHz)
00	15.75
01	7 875
10	5.25
11	3 9375

- メモリ

再生するデータをおくメモリを指定します。

メモリ	意味
0	メインメモリ
1	VRAM

HL　PCMデータのアドレス
VRAM指定時は、EレジスタとHLレジスタとをあわせて、3バイトで設定します。Eレジスタが最上位バイトです。

BC　PCMデータの長さ
VRAM指定時は、DレジスタとBCレジスタとをあわせて3バイトで設定します。Dレジスタが最上位バイトです。

戻り値　キャリーフラグ

値	意味
0	正常終了
1	異常終了 A　異常終了要因 1　サンプリング周波数指定誤り 2　[STOP] キーによる中断 HL　中断時のデータアドレス VRAM指定時は、EレジスタとHLレジスタとをあわせて、3バイトで返します。Eレジスタが最上位バイトです。

変更レジスタ　すべて

注　意　Z80 モードのときには、自動的に R800 ROM モードに切り換えて実行し、終わると Z80 モードに戻ります。
PCM を再生しているあいだは、割り込みは禁止されます。なお、STOP キーが押されると実行を中断します。
この BIOS は、実際にはページ 1（04000H～07FFFH）におかれたプログラムで実行されます。したがって、データをメインメモリにおくときは、必ず 08000H 番地以降でなければなりません。

PCMREC(0189H/MAIN)

機　能　PCM のデータを録音します。

コール手順　A

b7	b6	b5	b4	b3	b2	b1	b0
R	T	T	T	T	C	S	S

- S S（b1–b0）— サンプリングレート
- C（b2）— 圧縮
- T T T T（b6–b3）— トリガーレベル
- R（b7）— VRAM/メインメモリ

- サンプリングレート

  サンプリングレートを指定します。

サンプリングレート	意味 (KHz)
00	15.75
01	7.875
10	5.25
11	3.9375

- 圧縮

  圧縮するかしないかを指定します。

圧縮	意味
0	圧縮しない
1	圧縮する

- トリガーレベル

  録音のきっかけとなる音の大きさを指定します。

- メモリ

  録音するメモリを指定します。

メモリ	意味
0	メインメモリ
1	VRAM

HL　PCMデータのアドレス

　　VRAM指定時は、Eレジスタと HL レジスタとをあわせて、3バイトで設定します。Eレジスタが最上位バイトです。

BC　PCMデータの長さ

　　VRAM指定時は、Dレジスタと BC レジスタとをあわせて3バイトで設定します。Dレジスタが最上位バイトです。

戻り値　　キャリーフラグ

値	意味
0	正常終了
1	異常終了 A　異常終了要因 　1　サンプリング周波数指定誤り 　2　[STOP] キーによる中断 HL　中断時のデータアドレス 　VRAM 指定時は、Eレジスタと HL レジスタとをあわせて、3バイトで指定します。Eレジスタが最上位バイトです。

変更レジスタ　　すべて

注　意　　Z80モードのときには、自動的に R800 ROM モードに切り換えて実行し、終わると Z80 モードに戻ります。

Z80モードや R800ROMモードのときに、サンプリングレートを 15.75KHz に指定すると、「サンプリング周波数指定誤り」エラーになります。録音中は、割り込みは禁止されます。なお、[STOP] キーが押されると実行を

中断します。
メインメモリに録音するときは、PCMPLY と同じように、データを録音する番地に注意して下さい。

圧縮
再生時には、常にデータは圧縮されているものと見なして再生します。
録音時には、圧縮スイッチにより圧縮処理の有無を指定できます。ただし、録音データが0のときは、常に1に読みかえて記録します。

5.2 変更されたエントリ

MSX turbo R の仕様により、内容が変わったエントリもあります。
以下では、それぞれについて変更された点を説明します。

GTPDL(00DEH/MAIN)

機　能	パドル機能はなくなりました。常に0が返されます。

TAPION(00E1H/MAIN)

機　能	常にキャリーフラグを立て、エラーとしてリターンします。

TAPIN(00E4H/MAIN)

機　能	常にキャリーフラグを立て、エラーとしてリターンします。

TAPIOF(00E7H/MAIN)

機　能	常にキャリーフラグを立て、エラーとしてリターンします。

TAPOON(00EAH/MAIN)

機　能	常にキャリーフラグを立て、エラーとしてリターンします。

TAPOUT(00EDH/MAIN)

機　能　　常にキャリーフラグを立て、エラーとしてリターンします。

TAPOOF(00F0H/MAIN)

機　能　　常にキャリーフラグを立て、エラーとしてリターンします。

STMOTR(00F3H/MAIN)

機　能　　何もせずにリターンします。

GTPAD(00DBH/MAIN)

機　能　　ライトペン機能はなくなりました。A レジスタに 8～11 を入れてコールすると、常に 0 が返されます。

NEWPAD(01ADH/SUB)

機　能　　ライトペン機能はなくなりました。A レジスタに 8～11 を入れてコールすると、常に 0 が返されます。

RDRES(017AH/MAIN)

機　能　　システム予約です。

WRRES(017DH/MAIN)

機　能　　システム予約です。

6章 マッパーRAMセグメント

ここでは、R800 DRAMモードでシステムセグメントに割り付けられた、マッパーRAMセグメントを再利用するための手順を紹介します。

6.1 MSX turbo R のマッパーサポートルーチン

MSX turbo R のマッパーサポートルーチンは、R800 DRAMモードを考慮して、以下のように拡張されています。

1. 常に、内蔵スロット上のマッパーRAMをプライマリマッパーとして選択します。
2. プライマリマッパーRAMの最後の4セグメントを、R800 DRAMモード用のセグメントとして割り付けます。
3. DOS2カーネルのRAMセグメントは、R800DRAMモード用セグメントの前の2セグメントに割り付けます。

マッパーサポートルーチンでは、R800 DRAMモードであるかどうかに関わらず、実際にシステムに実装されているRAM容量をもとにして、管理するプライマリマッパーの総セグメント数を決めていることに注意して下さい。

本体のRAMが256KBの場合、初期化が終わった段階では、割り付けテーブルは以下のようになります。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	S	S	S	−	−	−	−	−	−	S	S	S	S	S	S

図 1.3 初期化時の割り付けテーブル

ここで、S とあるのは、「システムセグメントとして割り当てられている」という意味で 内訳は次のとおりです。

表 1.4 初期化時のシステムセグメント

セグメント番号	内容
0〜3	アプリケーションが使用する。
10〜11	MSX-DOS2 カーネルが使用する。
12〜15	割り当てられているが、MSX-DOS2 では使用しない。

この状態ではセグメント 4〜9 が割り付けの対象になります。

R800 DRAM モードのとき、最後の4セグメントはライトプロテクトされており、RAMとしてアクセスできないので、マッパーサポートルーチンが返す総セグメント数のうちの4セグメントは、RAMとして扱うことはできません。しかし、セグメント管理の立場からはこの4セグメントはシステムセグメントとして割り付けられており、アプリケーションに対する割り付けの対象にならないので、実質上の弊害はありません。

最後の4セグメントには、システムの初期化時に、システム ROM の内容がコピーされています。各セグメントにコピーされる ROM の内容は以下のとおりです。

表 1.5 コピーされるシステム ROM の内容

セグメント番号	内容
12	MAIN ROM（ページ0）
13	MAIN ROM（ページ1）
14	SUB ROM
15	漢字ドライバの前半 16KB

どのような場合にも、これらの4セグメントは常にマッパーサポートルーチンが管理しているので R800 ROM モードの場合には、標準的な手順の「セグメントの解放」ファンクションを使って、解放することができます。解放されたセグメントは、以降「セグメントの割り付け」ファンクションの対象になるので、RAM を有効に使うことができます。以上のように、マッパーサポートルーチンは、R800 ROM モードに切り換えた際に、必要に応じて最後の4セグメントを RAM セグメントとして再利用できるような構成になっています。

ただし、12〜15 のセグメントをアプリケーションなどで使用したときは、R800 DRAM モードに変えることはできません。どうしても R800 DRAM モードに変更したいときは、システム ROMの内容を、表 1.5 で示したセグメントにコピーし、それらのセグメントをシステムセグメントとして割り付けてからにして下さい。

6.2 最後の 4 セグメント利用の手順

以下で、最後の4セグメントを再利用するための、具体的な手順を説明します。

1. R800 DRAM モードであることを確認します。
   最後の4セグメントが初期化されたままの状態であり、他の目的で解放または再割り付けされていないことを確認しなければなりません。
2. 拡張 BIOS の「マッパーサポートルーチンアドレスの獲得」ファンクションを使って、プライマリマッパーの総セグメント数を得ます。
   最後の4セグメントのセグメント番号を獲得します。総セグメント数を Nとすると $(N-4)\sim(N-1)$ が解放の対象となるセグメントです。
3. マッパーサポートルーチンの FRE_SEG ファンクションを使って最後の4セグメントを解放します。
   この際、対象となるのはプライマリマッパーなので、Bレジスタは0です。

6.3 プログラム例

プログラム例を以下に示します。この例では、エラー処理や例外状態の判別は行なっていませんのでご注意下さい。また、jump_table は RAM 上になければなりません。

```
                .z80
extbio  equ     0ffcah          ;extended bios entry

        ld      de,0402h        ;
        call    extbio          ;マッパーサポートルーチンアドレスを獲得
        ld      de,jump_table   ;マッパーサポートルーチンの呼び出し効率
        ld      bc,16*3         ;を上げるため、固定アドレスにジャンプテー
        ldir                    ;ブルをコピーする。
        ld      b,4
fre_loop:
        dec     a               ;A = N-1, N-2 ..., N-4
        push    bc
        push    af
        ld      b,0             ;プライマリマッパーのセグメント [A] を解放
        call    fre_seg         ;する。
        pop     af
        pop     bc
        djnz    fre_loop
        ...

        jump_table:
        all_seg:        ds      3
```

```
fre_seg:    ds    3
rd_seg:     ds    3
wr_seg:     ds    3
cal_seg:    ds    3
calls:      ds    3
put_ph:     ds    3
get_ph:     ds    3
put_p0:     ds    3
get_p0:     ds    3
put_p1:     ds    3
get_p1:     ds    3
put_p2:     ds    3
get_p2:     ds    3
put_p3:     ds    3
get_p3:     ds    3
```

7章 新しいハードウェア

この章では、MSX turbo R に新しく内蔵されているハードウェアについて説明します。

7.1 システムタイマー

R800 は、命令の実行時間が一定ではないので、CPU の命令実行時間から、ウエイトのタイミングなどをとることができません。そこで、MSX turbo R では、数 10μ秒〜数 10m秒程度の短い時間の検出ができるように、システムタイマーが追加されました。

システムタイマーは、16 ビットのクリア・データ読み出しが可能なカウンタで、10.738635MHz のシステムクロックを 42 分周 (10.738635MHz/42=255.68KHz) したクロックでカウントアップされます。つまり、最下位ビットは 3.911μ秒毎にカウントアップされることになります。なお、カウンタをクリアしてからカウントが 1 になるまでの時間は、0〜3.911μ秒の範囲です。

カウンタの読み出し時に、データのラッチ機能はありません。カウンタの上位・下位読み出しの間にクロックが入ると、正しいカウント値が読み出されないので注意して下さい。

表 1.6 システムタイマーの I/O ポート

I/O ポート番地	R/W	機能
0E6H	W	任意のデータを書き込むと カウンタがクリアされる。
0E6H	R	カウンタデータの下位 8 ビットの読み出し。
0E7H	R	カウンタデータの上位 8 ビットの読み出し。

割り込みルーチンなどから呼ばれるプログラムは、カウンタをクリアすると、フォアグラウンドで動作しているプログラムに影響を与えるので、クリアしないで下さい。

具体的には、次のサンプルプログラムのようにします。

```
countlow        equ     0e6h    ; カウンタ下位8ビット
counthigh       equ     0e7h    ; カウンタ上位8ビット
        .z80
;
; Bレジスタで指定された時間待つ。
; 待つ時間は、B*3.911 μ秒。
; 誤差は、-3.911 μ秒～+0 μ秒。
; 0を指定してはいけない。
; 割り込みは禁止されていなければならない。
; C,A,Fの各レジスタは破壊される。
;
wait:
        in      a,(countlow)    ; カウンタの現在値を得る
        ld      c,a             ; それを保存する
wait_loop:
        in      a,(countlow)    ; カウンタの現在値を得る
        sub     c               ; 経過時間を算出する
        cp      b               ; 指定された時間経過したか?
        jr      c wait_loop     ; 経過していなければループする
        ret
```

7.2 PCM

ここでは、PCMのI/Oポートや録音・再生の手順などを解説します。

7.2.1 PCMのI/Oポート

PCM録音・再生のためのI/Oポートは以下のとおりです。

表1.7 PCMのI/Oポート

番地	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0A5H(Write)	0	0	0	SMPL	SEL	FILT	MUTE	ADDA
0A5H(Read)	COMP	0	0	SMPL	SEL	FILT	MUTE	BUFF
0A4H(Write)	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
0A4H(Read)	0	0	0	0	0	0	CT1	CT0

ADDA (BUFF)　　バッファモード

D/Aコンバータの出力をシングルバッファにするか、ダブルバッファにするかを指定します。D/A時にはダブルバッファに、A/D時にはシングルバッファにして下さい。

- 0 シングルバッファ (A/D時、リセット時)
- 1 ダブルバッファ (D/A時)

MUTE	ミューティング制御 システム全体の音声出力をON/OFFします。 0 音声出力OFF（リセット時） 1 音声出力ON
FILT	サンプル・ホールド回路入力信号の選択 A/D時にサンプル・ホールド回路に入力する信号を、フィルタの出力信号にするか、基準信号 (ground level) にするかを選択します。 0 基準信号（リセット時） 1 フィルタ出力信号
SEL	フィルタ入力信号の選択 ローパスフィルタに入力する信号を、D/Aコンバータの出力信号にするか、マイクアンプの出力信号にするかを選択します。 0 D/Aコンバータ出力信号（リセット時） 1 マイクアンプ出力信号
SMPL	サンプルホールド信号 入力信号をサンプルするか、ホールドするかを選択します。 0 サンプル時（リセット時） 1 ホールド時 A/D時には、信号をホールドする前に、最低7μ秒はサンプルしなければなりません。
COMP	コンパレータの出力信号 サンプル・ホールドの出力信号と、D/Aコンバータの出力信号とをくらべて、どちらが大きいかを読み出すことができます。 0 D/A出力>サンプル・ホールド出力 1 D/A出力<サンプル・ホールド出力
DA7~DA0	D/A出力データ
CT1、CT0	カウンタデータ 63.5μ秒ごとにカウントアップされます。 D/A時（ADDAが1）には、カウントアップに同期して、0A4H番地に書かれたデータが繰り返し出力されます。0A4H番地にデータを書き込むと、カウンタはクリアされます。 A/D時（ADDAが0）には、0A4H番地に書かれたデータはすぐに出力されます。0A4H番地にデータを書き込んでもカウンタはクリアされません。

7.2.2 PCM 再生

PCMは次の手順で再生します。

1. ADDA を 1 に、MUTE を 1 に、SEL を 0 にします (I/O ポートの 0A5H 番地に H00000011B を出力する)。
2. CT1、CT0 を読んで、サンプリング周期を検出します。
3. PCM データを出力します。このとき、カウンタは自動的にクリアされます。
4. データの個数回、2 と 3 とを繰り返します。

```
        .z80
;
; PCM 再生
;
; 入力    HL = PCM データの開始アドレス
;         BC = PCM データの長さ
;         E  = 再生サンプリング周期
;               1   15.75KHz
;               2   7.875KHz
;               3   5.25 KHz
;
; 戻り値  なし

pcmdac  equ    0a4h             ; D/A コンバータ  (Write)
pcmcnt  equ    0a4h             ; カウンタ         (Read)
pcmcntl equ    0a5h             ; PCM コントロール (Write)
pcmstat equ    0a5h             ; PCM ステータス  (Read)
pcmplay:
        ld     a,00000011b
        out    (pcmcntl),a      ; D/A モードに設定
        di                      ; タイミングの正確さのために
                                ; 割り込みを禁止する
pcmplay_loop:
        in     a,(pcmcnt)       ; カウンタを読み取る
        sub    e                ; 希望の値になったか?
        jr     c,pcmplay_loop   ; そうでないならループする
        ld     a,(hl)           ; データを読み取る
        out    (pcmdac) a       ; DAC に出力する
        inc    hl               ; 次のデータを指し示す
        dec    bc               ; カウンタを減らす
        ld     a,c              ; カウンタは 0 か?
        or     b
        jr     nz,pcmplay_loop  ; そうでないならループする
        ei                      ; 割り込み禁止を解除
        ret
```

7.2.3 PCM 録音

PCM 録音は次の手順で行います。

1. ADDA を 0 に、MUTE を 0 に、FILT を 1 に、SEL を 1 に、SMPL を 0 にします (I/O ポートの 0A5H 番地に 00001100B を出力して、入力アナログ信号をサンプルする)。
2. CT1、CT0 を読んで、サンプリング周期を検出します。
3. 最低 7μ秒待ちます。この 7μ秒が 2 に含まれていれば待つ必要はありません。
4. SMPL を 1 にして (I/O ポートの 0A5H 番地に 00011100B を出力する)、入力アナログ信号をホールドします。
5. 逐次変換のシーケンスによって、D/A コンバータのデータを上位ビットから変化させながら、D/A コンバータ出力と入力アナログ信号との比較結果を COMP から読み込んで、各ビットを決定し、データを格納します。
6. SMPL を 0 にします (I/O ポートの 0A5H番地に 00001100B を出力する)。
7. 2 から 6 を繰り返します。

```
        .z80
;
; PCM 録音
;
; 入力    HL = PCM データの開始アドレス
;         BC = PCM データの長さ
;         E  = 再生サンプリング周期
;              1   15.75 KHz
;              2    7.875KHz
;              3    5.25 KHz
; 戻り値  なし

pcmdac  equ     0a4h            ; D/A コンバータ
pcmcnt  equ     0a4h            ; カウンタ
pcmcntl equ     0a5h            ; PCM コントロール
pcmstat equ     0a5h            ; PCM ステータス
; 1 ビット A/D 変換マクロの定義
adconv  macro   next_bit,strip
        local   adconv_not_change
        out     (pcmdac),a      ; データを出力
        db      0edh 70h        ; PCMSTAT からデータを読み込み
                                ; フラグにのみ反映させる
        jp      m,adconv_not_change ; 入力アナログ信号の方が大きければ
                                ; データはそのまま
        and     strip           ; 入力アナログ信号の方が小さいので
                                ; ビットを 0 にする
adconv_not_change:
        or      next_bit        ; 次のビットを 1 にする
```

```
        endm

pcmrec:
        ld      a,00001100b
        out     (pcmcntl),a     ; A/D モードに設定
        ld      d,0             ; カウンタ検出の初期値を設定する
        di                      ; タイミングの正確さのために
                                ; 割り込みを禁止する
pcmrec_loop:
        in      a,(pcmcnt)      ; カウンタを読み取る
        cp      d               ; 希望の値になったか?
        jr      nz,pcmrec_loop  ; そうでないならループする
        add     a,e             ; 次のカウンタの値を作る
        and     11b
        ld      d,a             ; それを保存する
        ; 上記ループにより７μ秒を満足する
        exx
        ld      a,00011100b
        out     (pcmcntl),a     ; データをホールドする
        ld      c,pcmstat       ; COMP ビットのポートアドレスを設定する
        ld      a,80h           ; 初期データを設定する
        adconv  01000000b,01111111b  ; 順次ビットを変換する
        adconv  00100000b,10111111b
        adconv  00010000b,11011111b
        adconv  00001000b,11101111b
        adconv  00000100b,11110111b
        adconv  00000010b,11111011b
        adconv  00000001b,11111101b
        adconv  00000000b,11111110b
        exx
        ld      (hl),a          ; データを格納する
        ld      a,00001100b
        out     (pcmcntl),a     ; データのホールドを解除する
        inc     hl              ; 次のデータを指し示す
        dec     bc              ; カウンタを減らす
        ld      a,c             ; カウンタは０か?
        or      b
        jr      nz,pcmrec_loop  ; そうでないならループする
        ld      a,00000011b
        out     (pcmcntl),a     ; D/A モードに設定。MUTE を解除する
        ei                      ; 割り込み禁止を解除
        ret
```

注　意　　このプログラムは、Z80 では速度が間に合わない場合があります。

8章 アプリケーション作成上の注意

MSX turbo R で、アプリケーションプログラムを作成するときは、以下の点に注意して下さい。

8.1 MAIN ROM のバージョン番号

MAINROM の 002DH番地に入っているシステム（BASIC）のバージョン番号は以下のとおりです。

表 1.8 MAIN ROM のバージョン番号

機種	BASIC のバージョン番号	002DH 番地の内容
MSX	BASIC 1.xx	00H
MSX_2	BASIC 2.xx	01H
MSX_{2+}	BASIC 3.xx	02H
MSX turbo R	BASIC 4.xx	03H

MSX turbo R であるかどうかは、MAIN ROM の 002DH 番地の内容が、03H 以上であるかどうか（≧3）で判断して下さい。決して、3 であるかどうか（=3）で調べてはいけません。

8.2 MSX_{2+} および MSX turbo R で動作するソフトウェア

MSX_{2+}（または MSX_2）と MSX turbo R との両方で動作し、MSX turbo R のときは R800 モードで動かしたい MSX DOS1 のアプリケーションは、動作しているシステムが MSX turbo R であることを確認したら、BIOS の CHGCPU をコールし、CPU を R800 に切り換えることができます。

ただし、このアプリケーションがMSX DOS1のファンクションを呼び出すときには、CPUをZ80に切り換えてから呼び出すようにして下さい。MSX DOS1はR800のスピードに対応していないので、R800でMSX DOS1のファンクションを実行すると、ファイルを壊してしまう可能性があります。

また、基本的にこのようなアプリケーションは、MSX-DOS1やBASICのコマンド待ちの状態に返ってはいけません。R800とMSX-DOS1とを組み合わせたときの動作は、保証されていません。どうしてもアプリケーションを終了したいときには、MAIN ROMの0番地にジャンプして下さい。MSX DOS1やBASICのコマンド待ちに戻ることが必要なアプリケーションは、戻ったときには必ずZ80が選択されているようにして下さい。

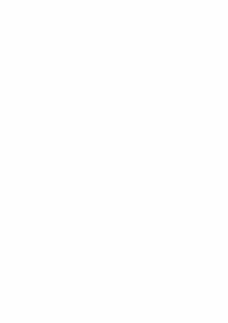
Z80はIX・IYレジスタの8ビットアクセスを保証していませんが、これを使用しているソフトウェアが多くみられます。R800ではこれを保証しているので問題はありませんが、以下のような場合にZ80とは異なった動作をします。

例えば

```
        LD      IXH IXL
```

に対する正しいオブジェクトはDD 65ですが、アセンブラによっては、DD DD 65というオブジェクトを生成するものがあります。

このオブジェクトはZ80では偶然に正しく動作するようですが、R800では動作が保証されていません。十分にご注意下さい。

第2部
MSX-DOS2

1章
MSX-DOS2とは

MSX-DOS version 2 (以下MSX-DOS2またはMSX-DOS) はMSX-DOS version 1.xx (以下MSX DOS1) と同様、複数のディスクファイルで供給されます。ディスクファイルはMSXDOS2.SYS、COMMAND2.COM、およびヘルプファイル、外部コマンドからなります。

MSXD●S2.SYSは機能が強化されたCP/Mコンパチブルな環境で、プログラムをロードおよび実行します。COMMAND2.COMは特殊なプログラムで、MS-DOSやMSX-DOS1のものと同様の機能があり、しかも拡張されたメモリの管理を実現するなど、より優れた数々の高度なコマンドや機能を提供します。

また、MSX DOS2はMSX DOS1用に記述されたプログラムや大部分の標準的なCP/Mのプログラムをロード・実行することができ、引数を使用するバッチファイルや、MS-DOS 2.11互換ファイルシステムをサポートし、MS DOSと同様の他の機能を実現します。

さらに、APPEND機能が提供されており、階層ディレクトリを扱うように作成されていないCP/Mのプログラムで、階層ディレクトリが簡単に使用できます。

2章 コマンド行の編集

MSX-DOS においてコマンド行で入力を行う場合には、入力ミスの訂正や以前のコマンドの再入力および編集といった、簡単な編集機能が使用できます。キーボードで通常の文字を入力すると、それらの文字はそのまま画面上に表示されます。大部分のコントロール文字を入力すると、それらは「^」記号に続くコントロール文字によって表現されます。例えば CTRL + A は^A と表示されます。例外は、以下のコントロール文字です。

表 2.1 コントロール文字

特殊キー	機能
CTRL + M、↵	コマンドの実行を開始します。コマンド行の入力、編集が完了したときに押します。
CTRL + H、BS	カーソルのすぐ左にある文字を削除することができます。コマンド行で入力しているときはいつでも使うことができます。
CTRL + I、TAB	タブ
CTRL + R、INS	「上書モード」と「挿入モード」を切り換えます。上書モードのとき、カーソルの形状は1文字分の大きさの矩形で入力された文字はカーソルの下にある文字を重ね書きします。挿入モードのとき、カーソルの形状は1文字の半分の大きさの矩形で、入力された文字はカーソルの前に挿入されます。
CTRL + [、ESC	行をクリアし、新しい行の入力ができるようにします。
CTRL + X、SELECT	〃
CTRL + K、HOME	カーソルを行の先頭に移動します。
CTRL + C	ブレークキーとして機能します。より効果的で望ましいブレークキーは CTRL + STOP です。

特殊キー	機能
CTRL + J	改行コードですが、コマンド行で入力されたときは何もおきません。
CTRL + N	CTRL + P によってオンにされたプリンタエコーをオフにします。
CTRL + P	プリンタエコーをオンにします。オンにすると、画面上に表示されるすべての文字がプリンタにも送られます。
CTRL + ■	他のキーが押されるまで、すべての文字の出力を停止します。
CTRL + U	現在入力中の行を消去します。

その他の編集機能は以下のとおりです。

表 2.2 編集機能

特殊キー	機能
DEL	カーソルの下にある文字を削除します。
←、→	カーソルを行の中で左右に移動させます。
	カーソル移動後に文字を入力すると、上書きモードの時は、その時点でカーソルの下にある文字に重ね書きし、挿入モードの時は、その時点でカーソルの前に入力した文字が挿入されます。

また、コマンド行エディタのリストは以前に入力されたコマンドのリストを最大256文字まで保存しています。↑ キーを押すと、そのリストをさかのぼって、以前に入力されたコマンド行が表示され、このコマンド行を編集したり再入力したりすることができるようになります。同様に、↓ キーを押すと、その次に入力されたコマンド行に移動します。

以前のコマンド行が変更されると、それは新しいコマンド行として、リストの最後に追加されます。変更されずに入力された場合は、その行はリストに追加されず、現在行がその行に移動します。これによって、以前のコマンドのすべてのシーケンスが簡単に入力できるようになります。

リストは実際にはリング状構造になっており、リストの一番上あるいは一番下を超えて移動すると、リスト中の最後あるいは最初のコマンドにそれぞれ移動します。

ここで記述された機能は、実際にはMSX-DOS上で実行される多くのプログラムから利用できます。MSX DOSの「バッファ行入力」ファンクションコールを実行するプログラムでは、上に述べられたように以前の行を呼び出して再入力したり編集したりすることができます。もちろん以前の行には以前のコマンドも含まれます。

3章
バージョンアップにともなう変更点

MSX-DOS2は何度か拡張、改良され現在次のようなのバージョンがあります。

表 2.3 MSX DOS2 のバージョン

バージョン番号		内容
MSX DOS version	2.20	日本語 MSX-DOS2（アスキー発売）
	2.30	MSX turbo R FS A1ST（Panasonic製）内蔵
	2.31	MSXturbo R FS-A1GT（Panasonic製）内蔵

VER コマンドにより、現在お使いのMSX-DOS2のバージョンを確認することができます。
また、「カーネル」、「MSXDOS2.SYS」、「COMMAND2.COM」のバージョンはそれぞれ異なっていても使用できます。
この章では、バージョンアップにより拡張した機能について説明します。

3.1 version 2.30 の新機能

3.1.1 カーネルの変更

- MSX turbo R 専用です。MSX_{2+}以前の機種では動作しません。
- プライマリマッパは必ず内蔵 RAM から選びます。また RAM ターボモードで使用する RAM セグメントの最後の 4 ページはシステムセグメントとしてリザーブします。
- 新たにファイルをオープンするときに、既にオープン済みのファイルについては、ボリューム ID もチェックします。途中でディスクが交換されたら、同ドライブ、同ディレクトリエントリ、同名のファイルでも別ファイルとして認識します。ただし、これは MSX DOS2 フォーマットのディスクに限ります。

3.1.2 COMMAND2.COM の変更

- バッチ終了時に「SET ECHO OFF」を行わないようにしました。これまではバッチ終了時に「SETECHOOFF」を行っていました。
- 1画面ごとの出力停止（/P）は、これまで論理行をカウントしていましたが、表示行をカウントするようにしました。表示行がラップしたときも行数にカウントします。
- PAUSE コマンドのプロンプトが短くなりました。SCREEN 1のデフォルト幅（WIDTH 29）で1行におさめるためです。

旧

```
Press any key to continue...
```

新

```
Press any key to continue
```

- 環境変数 KHELP を追加しました。HELP コマンド実行時に、画面モードに応じて ANK モードならば HELP で指定されるディレクトリを、漢字モードならば KHELP で指定されるディレクトリを自動的に選択します。それぞれデフォルトは MSX-DOS2 が起動されたドライブのルートディレクトリ中の HELP、KHELP というディレクトリになっています。
- 環境変数 EXPERT を追加しました。EXPERT はMSX-DOS1 でフォーマットされたディスク上のプログラムを実行させるかさせないかを制御します。これは MSX-DOS のバージョンの違いからくる問題を未然に防ぐために追加しました。「ON」以外の値はすべて「OFF」として解釈します。EXPERT が存在しない（デフォルト）か「OFF」の場合は MSX-DOS1 でフォーマットされたディスクからのプログラムの実行は禁止されます。この場合、次のようなプロンプトが出力されます。

```
*** Wrong version of MSX-DOS
```

```
*** MSX-DOS のバージョンが違います
```

EXPERT が「ON」の場合は、MSX-DOS1 でフォーマットされたディスクからのプログラムの実行が可能になります。

3.1.3 外部コマンドの変更

- CHKDSK コマンド
  メディアタイプが MSX-DOS2 フォーマットでない場合その旨を表示します。
- DISKCOPY コマンドに/S スイッチを追加しました。
  /S が指定されると、ブートコードもコピーします。

3.1.4 その他の変更

- Z80 および R800 モードの判別（カーネルの変更）
  ブートシーケンスには以下の項目を追加してモードの設定をしています。
  a. シフト立ち上げのとき、[1] キーが押されていなければ R800 モード
  b. [1] キーが押されていれば Z80+MSX-DOS1
  c. H.STKE がセットされていたとき
     USRTAB が FCERR を指しているならば、Disk-BASIC1
     USRTAB が FCERR を指していないならば、Disk BASIC2
  d. ブートメディアが MSX-DOS1 のときは、MSX-DOS1

なお、ディスクドライバは Z80 モードで呼びます。これは、タイミングに依存するディスクドライバとの互換性のためです。

3.2 version 2.31 の新機能

3.2.1 カーネルの変更

- ファイル名は、漢字モード、ANK モードにかかわらず、シフト JIS として解析します。これまでは漢字モードのときのみシフト JIS として解析していました。「パス名の解析」(05BH、_PARSE)、「ファイル名の解析」(05CH、_PFILE)、「文字の検査」(05DH、CHKCHR) のファンクションは、言語設定にかかわらずシフト JIS 固定になりました。

3.2.2 COMMAND2.COM の変更

- 環境変数名の前後に%をつけることで、コマンド行に環境変数を取り込めるようにしました。

```
A>ECHO %PATH%
; A:¥UTILS
```

環境変数 PATH の内容を表示します。

- コマンド IF を追加しました。

4章 MSX-DOS2への移植の注意

この章では、MSX-DOS1上でプログラムを作成する実力を持った方を対象にして、MSX-DOS2の機能を十分に生かしたプログラムを作る際の注意点について解説します。

4.1 ファイルハンドルの利用

ファイルにアクセスするには、MSX-DOS1のときにはFCBを使いましたが、MSX-DOS2ではこの他にファイルハンドルを使うことが可能です。ファイルハンドルを使ったアクセスには以下のような利点がありますので、これを使うことを強くお奨めします。

1. カレントディレクトリ以外のファイルを扱える。
2. FCBを保持するには37バイト必要だが、ファイルハンドルを保持するには1バイトでよい。
3. ファイル名を「8バイトの主ファイル名と3バイトの拡張子」に整形する必要がない。

従来のFCBを使ったファイルI/Oは、以下のような手順を踏んでいました。

1 FCBを用意する。
2 _FOPEN (0FH) または_FMAKE (016H) を呼ぶ。
3 ブロックサイズを設定する (普通は1)。
4 FCBのランダム・レコード・フィールドを設定する。
5 _SETDTA (01AH) で転送アドレスを設定する。
6 RANDOM READ,WRITE (_RDBLK 027H、_WRBLK 026H) を呼ぶ。
7 _FCLOSE (010H) を呼ぶ。

これに対してファイルハンドルを使ったI/Oは以下のようになります。

1' パス名（ASCIIZ文字列:ヌル文字で終了する文字）を用意する。
2' _OPEN（043H）またはCREATE（044H）を呼んでハンドル番号をもらう。
4' _SEEK（04AH）を呼んで読みたい・書きたいバイトの所へ移動する。
6' _READ・_WRITE（048H・049H）を呼ぶ。
7' _CLOSE（045H）を呼ぶ。

ファイルハンドルを使う場合は3'や5'はなくなっています。これは、ブロックサイズは1に固定され、転送アドレスは_READ、_WRITE時に指定されるためです。

FCBによるI/Oと違って、ファイル名は8+3のフォーマットに直す必要はありません。またファイルハンドルを得たら以後パス名は不要です。プログラムはファイルハンドル番号（1バイト）だけを記憶していれば、その後の処理ができるようになっています。

ASCIIZで書かれたファイル名を8+3のフォーマットに変換するには_PFILE（05CH）が便利です。「.」、「..」や「*」などの扱いも行います。

個々のファンクションの詳細は、17.3「ファンクションの説明」を参照して下さい。

4.2 漢字を扱う際の注意点

漢字の内部表現はシフトJISです。シフトJISは第1バイトが080H~09FH 0E0H~0FCHで、第2バイトが040H~07EH、080H~0FCHの範囲にあります。注意しなければならないのは第2バイトが、他の1バイトコード（040H~07EH、0A0H~0DFH）や漢字の第1バイトの範囲と重なっていることです。その結果、1バイトを見ただけでは、それがどんな文字か判らないということになります。

00H~03FH、および07FHは、漢字の第1バイト、第2バイトのどちらとも重なっていないので、1バイト見ただけで漢字ではないと判断できます。しかし、それ以外の場合は漢字かどうかを判別することはできません。

つまり、どんな文字かを知るためには、00H~03FH、07FHが見つかったところから、順番にチェックしていかなければなりません。例えば漢字テキストファイルのある場所からファイルの先頭に向かって、ある文字を検索する、などの作業は決して簡単なものではありません。

単に現在のスクリーンモードがANKモードか漢字モードかを調べるには、拡張BIOSコールを使います。A=0、D=011H E=0にして0FFCAHを呼び、A=0で戻ってきたらANKモードです。ただし、このときSPはページ1以外になければなりません。

文字の属性などを調べるには_CHKCHR（05DH）のファンクションコールを使います。調べる文字列内の文字を順番にこのファンクションに渡すことにより、文字の種類を判別できます。ANKモードか漢字モードかの判断は、このファンクションの中で行われるので、呼び出す側はモードを調べる必要がありません。

漢字を含むパス名やファイル名を解析するのには_PARSE（05BH）、_PFILE（05CH）を使います。漢字の第2バイトが「¥」（05CH）である場合でも、このファンクションが適切

な処理を行います。ANK モードか漢字モードかの判断は、このファンクションの中で行われるので、呼び出す側はモードを調べる必要がありません。

4.3 I/O コントロールの利用

アプリケーションによっては、スクリーンのサイズや、出力がスクリーンにいくのかファイルまたはデバイスにいくのかが問題になる場合があります。_IOCTL (04BH) はこのためのファンクションで、今まで 0F3B0H (LINLEN) を見て判断していたプログラムは、このファンクションを使うように変更することが望ましいでしょう。

4.4 環境変数の利用

アプリケーションによっては、環境変数を使うことによって操作性や性能が大幅に向上するものがあります。例えばテンポラリファイルを作成するディレクトリの指定などがあります。環境変数を扱うファンクションには、_GENV (06BH)、_SENV (06CH)、_FENV (06DH) があります。

環境変数それ自体はアプリケーションから直接見える空間にはなく、カーネルのデータセグメントにあります。これらのファンクションは、環境変数とアプリケーション空間との橋渡しをするものです。

5章 コマンド

この章はMSX DOS2に標準で含まれるすべてのコマンドを詳細に説明します。それぞれのコマンドは表記法にしたがって説明します。

5.1 この章の表記法

MSX-DOSで利用できるコマンドの構文の説明には、以下のような表記法を使用して説明します。

- 大文字の単語
  キーワードであり、示されたとおりに入力しなければなりません。しかし、大文字・小文字は区別されないので、これらを混ぜて使用してもかまいません。
- 日本語の項目
  コマンド行中のその位置でコマンドに与えなければならないパラメータです。
- 角形かっこ([])で囲まれた項目
  省略可能な項目です。角形かっこ自体をコマンド行に含めてはいけません。
- 縦棒(|)で区切られた項目
  項目のうちひとつを選択する必要があることを示します。縦棒自体はコマンド行に含めてはいけません。
- 枠で囲まれたテキスト
  画面の表示例であることを示します。

以下に示すのは、コマンド行上に指定することのできる項目です。

- d

  ドライブ名が必要であることを示します（A:、B:など）。

  d:が省略可能であるとき、指定されなければカレントドライブが使用されます。カレントドライブは、コマンドプロンプトによって示されます。

- パス

  ディレクトリパス名が必要であることを示し、その構文はMS DOSのものと同様です。パス名中のそれぞれのディレクトリは「¥」記号によって区切られます。パス名の先頭に「¥」記号がある場合はパス名がルートディレクトリから始まることを示し、そうでない場合は、パス名はCHDIRコマンドによって示されるカレントディレクトリから始まることを示します。ファイル名がパス名の後に続く場合は、パス名とファイル名は「¥」記号によって区切らなければなりません。2つの連続したドット「..」はパス名中ですぐ上の親ディレクトリを表します。単一のドット「.」はパス名中でカレントディレクトリを表し、したがってそれは通常パス名指定において何の意味も持ちません。

  海外仕様のMSXマシンでは、パス名の区切り記号は「¥」ではなく、バックスラッシュ（「＼」）が表示されます。

  書式でパス名が省略可能として示され、指定されなかった場合には、カレントディレクトリが使用されます。カレントディレクトリは、CHDIRコマンドで示されます。

  パス名を構成するディレクトリ名の構文は、以下に示されるファイル名の構文にしたがいます。

- ファイル名

  ファイルの名前が必要なことを示します。

  ファイル名は以下の構文をとります。この構文はMS DOSおよびMSX-DOS1と同様です。

  主ファイル名[.拡張子]

  ここで主ファイル名とは8文字までの文字列であり、拡張子は3文字までの文字列です。この制限を超える文字はすべて無視されます。主ファイル名または拡張子の中でワイルドカード文字を使うことができます。

  ワイルドカードとはファイル名を指定する際に、任意の1字または文字列に対応してその文字または文字列の代わりに用いることができる省略記号のことです。

  ワイルドカード文字を用いると、ファイルを指定する際にいくつかのファイルをまとめて指定することができます。

  ワイルドカード文字には、任意の1文字に対応する「?」（クエスチョンマーク）と任意の文字列に対応する「*」（アスタリスク）の2種類があります。

拡張子を与える場合には、それは1つのピリオド「.」によって主ファイル名の部分から区切らなければなりません。以下の文字はファイル名の中で使用することはできません。

　　コントロールコードおよびSPACE（文字コード0H~20Hおよび7FH、FFH）

　　:　;　.　,　=　+　¥　<　>　|　/　"　[　]

ファイル名として与えられた文字は大文字に変換されます。したがって大文字と小文字は同じ意味を持つことになります。2バイトの漢字コード（シフトJISコード）を使用することもできます。

ファイル名が省略可能である場合にそれが指定されないと、ファイル名「*.*」が使用されます。

- ファイルスペック

　これは、ディスク上の同一のディレクトリ中の特定のファイルあるいはいくつかのファイルを識別するために使用されます。構文を次に示します。

　　[d:][パス][ファイル名]

　ここで、3つの省略可能な項目のうち最低ひとつは指定されなければなりません。これが存在するファイルを列挙するため使用される場合には、/Hスイッチを指定することで不可視ファイルを識別することができます。

　一般に、d:が指定されないと現在のカレントドライブが使用され、パスが指定されないとそのドライブのカレントディレクトリが使用され、ファイル名が指定されないとファイル名「*.*」が使用されます。

- 複合ファイルスペック

　これは、コマンドが適用されるファイルやディレクトリを指定するために使用されます。構文を次に示します。

　　ファイルスペック [+ファイルスペック [+ファイルスペック...]]

　すなわち、いくつかのファイルスペックを「+」記号によって区切って指定できます。また、その際に、+の前後にスペースが入ってもかまいません。コマンド中でのこの機能は、すべての適合するファイルが単一のファイルスペックによって指定されたのとまったく同じです。

　複合ファイルスペックが存在するファイルを指定するために使用された場合、/Hスイッチ（上記「ファイルスペック」参照）をそれぞれのファイルスペックの後に指定することができます。この場合、/Hスイッチは指定したファイルスペックにのみ働きます。複合ファイルスペックの前に/Hスイッチが指定された場合は、すべてのファイルスペックに対して指定したことになります。

- ボリューム名

これはボリューム名が必要であることを示します。ボリューム名は最大、半角で11文字、全角で5文字の文字列であり、コントロールコードと「/」を除いて、ファイル名としては無効な文字も含めることができますが、先頭に空白を入れることはできません。

- デバイス

これはMSX-DOSの5つの標準デバイスのひとつが必要であることを示します。標準デバイスとその意味を以下に示します。

デバイス名	意味
CON	スクリーン・キーボード入出力
NUL	ヌルデバイス（何もしない）
AUX	補助入出力（例えば RS-232C シリアルポート）
LST	プリンタ出力
PRN	プリンタ出力

デバイス名の後にコロンは必要ありません。

デバイス名は通常ファイル名が使用できる場合には常に使用できます。例えば、コマンド COPY MYFILE PRN はファイル MYFILE を読んで、それをプリンタに書き出します。

CON デバイスを入力ファイル名として使用すると、行はコマンド行と同一の方法で入力、編集ができます（2章「コマンド行の編集」を参照）。処理を終了するには、[CTRL]+[Z]（^Z）を行の先頭で入力します。例えば、MYFILE という小さなテキストファイルを作成したい場合には、コマンド COPY CON MYFILE で作成できます。

```
A>COPY CON MYFILE
All work and no play makes Jack a dull boy.
Can you hear me ?
^Z
A>
```

上記の例のように、コンソールからテキストの行が入力でき、入力されたテキストは MYFILE というファイルに書き出されます。コマンドは単一の [CTRL]+[Z] を持つ行が入力されたときに終了します。

NUL デバイスに書き込んだ場合は、書き込まれた文字は単に無視されます。また NUL から読み出すと、エンドオブファイルが直ちに返されます（上記の例で [CTRL]+[Z] を入力したのと同じ）。

大部分のコマンドでは、デバイスを指定するのは意味がありません（例えば、CON デバイスは DEL コマンドで削除できません）。デバイスが使用されるようなコマンドは CONCAT、COPY、TYPE のようなファイルとデータのやり取りを行うものです。

- 数値

  これは数値が必要であることを示します。コマンドによって0~255まで、あるいは0~65535までの範囲をとります。

- セパレータ

  この記法を使って複数のパラメータが記述されている場合、それらはセパレータによって区切られなければなりません。セパレータは、0個以上の先頭の空白、区切り文字そして0個以上のそれに続く空白によって構成されます。使用できる区切り文字を次に示します。

  空白　タブ　;

  「/」文字で始まるスイッチ文字はこの例外であり、セパレータが前にくる必要はありません。

MSX DOSあるいはCP/M-80のプログラムはファイル名の主ファイル名と拡張子「COM」(指定しなくてもよい)を入力することによってロード、実行できます。バッチファイルも拡張子が「BAT」であることを除いて、同様に実行することができます。COMおよびBATファイルが同じ名前で同一のディレクトリに存在する場合には、COMファイルがBATファイルよりも先に見つけられて実行されます。コマンドのディスク上での正確な位置はその名前にそのドライブまたはパスを付けることによって指定できます。この場合、指定されたドライブの指定されたディレクトリが検索され、見つからなかった場合は「Unrecognizedcommand」(漢字メッセージの場合は、「コマンドが違います」)エラーとなります。

ファイル名と拡張子(指定しなくてもよい)だけが与えられると、まずカレントドライブのカレントディレクトリが捜されます。そこで見つからない場合、PATHのディレクトリリストで指定されたディレクトリを順番に検索します。このリストはPATHコマンドによって指定、あるいは変更することができます。それでもなお見つからない場合には、「Unrecognized command」エラーとなります。

CP/Mでは、ディレクトリやパス名が存在しないため、CP/Mのプログラムはこれらを指定することができず、それぞれのドライブのカレントディレクトリしかアクセスできません。これらのプログラムを使いやすくするために環境変数APPENDが使用できます。これにより、カレントディレクトリと同じように別のディレクトリを検索することができます。

ほとんどのコマンドやプログラムは、「標準入力」と「標準出力」を使用して、入力あるいは出力を行います。標準入力は通常キーボードに割り当てられ、標準出力は通常画面に割り当てられていますが、コマンド行にリダイレクション記号(「<」、「>」および「>>」)を書き、その後にデバイスまたはファイル名を指定することでそのコマンドを実行している間別のものに変えることができます。ひとつのコマンドの標準出力はまた、2つのコマンドを「|」記号でつなぐことによって、次のコマンドの標準入力に送ることができます。これらの機能についての詳細は、6章「リダイレクションとパイプ」を参照して下さい。

外部コマンドが実行されると、そのコマンドはCOMMAND2.COMが使用していたメモリの一部をオーバーライトすることがあります。したがってコマンドが終了すると、COM-

MAND2.COM は最初にロードされた COMMAND2.COM ファイルからメモリへ自分自身を再ロードする必要があることがあります。このファイルは SHELL の環境変数で指定されるファイルが検索され、そこになかった場合にはブートドライブのルートディレクトリの COMMAND2.COM が検索されます。それでも見つからないと、メッセージが出力されます。例えば、MSX-DOS がドライブ A からブートされたとすると、メッセージは次のようになります。

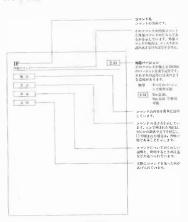
```
Insert COMMAND2.COM disk in drive A:
Press any key to continue
```

```
COMMAND2.COM の入ったディスクをドライブ A:に入れて
何かキーを押して下さい.
```

ルートディレクトリに COMMAND2.COM があるディスクをドライブ A に挿入してキーを押すと，COMMAND2.COMが再ロードされ、システムの動作が正常に続行します。

~~以上のようなコマンドとは意味あいが異なりますが、現在のカレントドライブは次のようなコマンドを指定することによって変更することができます。~~

コマンド名
コマンドの名前です。
そのコマンドが内部コマンドと外部コマンドのどちらであるかを示しています。外部コマンドの場合は、ディスクから読み込まなければなりません。
IF
内部コマンド
2.31
対応バージョン
そのコマンドが使えるDOS2のバージョンを表す記号です。それぞれの記号には次のような意味があります。
無印　すべてのバージョンで使用可能
2.31　Ver.2.30、Ver.2.31で使用可能
機能
書式
解説
文例
コマンドの内容を簡単に説明しています。
コマンドの書き方を示しています。<>で囲まれた項目は、何らかの数値や文字を指定し、[ ]で囲まれた項目は、省略可能であることを示します。
コマンドについてのくわしい説明と、使用するときの注意などが述べられています。
実際にコマンドを使った例があげられています。

図 2.1 この章の表記法

5.2 コマンド一覧

以下に示すのは、MSX-DOS で利用できる標準の全コマンドのリストです。その構文と目的も記します。

- ASSIGN [d: [d:]]
  論理ドライブと物理ドライブとの対応を設定します。
- ATDIR +H|–H [/H][/P] 複合ファイルスペック
  ディレクトリの属性を変更して、それらを可視・不可視にします。
- ATTRIB +H|–H|+R|–R [/H][/P] 複合ファイルスペック
  ファイルの属性を変更して、それらを可視・不可視に、そして読出し専用・読み書き可能に設定します。
- BASIC [プログラム]
  制御を MSX-Disk BASIC に渡します。
- BUFFERS [数値]
  システムのディスクバッファの数を表示・変更します。
- CD [d:][パス]
  カレントディレクトリを表示・変更します。
- CHDIR [d:][パス]
  カレントディレクトリを表示・変更します。
- CHKDSK [d:][/F]
  ディスク上のファイルの整合性を検査します。
- CLS
  画面をクリアします。
- COMMAND2 [コマンド]
  コマンドインタープリタを起動します。
- CONCAT [/H][/P][/B][/V][/A] 複合ファイルスペックファイルスペック
  ファイルを連結します。
- COPY [/A][/H][/T][/V][/P][/B] コピー元ファイル [コピー先ファイル]
  ファイルまたはデバイスから他のファイルまたはデバイスへデータをコピーします。
- DATE [日付]
  現在の日付を表示・設定します。

- DEL [/H][/P] 複合ファイルスペック
  ひとつあるいは複数のファイルを削除します。

- DIR [/H][/W][/P] [複合ファイルスペック]
  ディスク上のファイルの名前を表示します。

- DISKCOPY [d: [d:]][/X][/S]
  ひとつのディスクを別のディスクにコピーします。

- ECHO [テキスト]
  バッチファイル中でテキストを表示します。

- ERA [/H][/P] 複合ファイルスペック
  ひとつあるいは複数のファイルを削除します。

- ERASE [/H][/P] 複合ファイルスペック
  ひとつあるいは複数のファイルを削除します。

- EXIT [数値]
  COMMAND2.COM を終了し、呼び出したプログラムに戻ります。

- FIXDISK [d:][/S]
  ディスクを MSX-DOS2 フォーマットに更新します。

- FORMAT [d:]
  ディスクをフォーマット（初期化）します。

- HELP [項目]
  MSX-DOS 機能についてのオンラインヘルプを提供します。

- IF [NOT] 条件 コマンド
  条件判断をしてコマンドを実行します。

- [数値|OFF]
  KMODE [数値]—OFF [/S]
  漢字モードを設定・解除します。

- MD [d:] パス
  新しいサブディレクトリを作成します。

- MKDIR [d:] パス
  新しいサブディレクトリを作成します。

- MODE 数値
  画面の桁・行数を変更します。

- MOVE [/H][/P] 複合ファイルスペック [パス]
  ディスク上で、ファイルを別のディレクトリに移動します。

- MVDIR [/H][/P] 複合ファイルスペック [パス]
  ディスク上で、サブディレクトリを別のディレクトリに移動します。
- PATH [[+ | -][d:] パス [[d:] パス [[d:] パス...]]]
  COM および BAT コマンド検索パスを表示・変更します。
- PAUSE [コメント]
  バッチファイル中で、プロンプトを出してキーが押されるのを待ちます。
- RAMDISK [数値 [K]][/D]
  RAM ディスクのサイズを表示、あるいは設定します。
- RD [/H][/P] 複合ファイルスペック
  ひとつあるいは複数のサブディレクトリを削除します。
- REM [コメント]
  バッチファイル中にコメントを入れます。
- REN [/H][/P] 複合ファイルスペック ファイル名
  ひとつあるいは複数のファイルの名前を変更します。
- RENAME[/H][/P] 複合ファイルスペック ファイル名
  ひとつあるいは複数のファイルの名前を変更します。
- RMDIR [/H][/P] 複合ファイルスペック
  ひとつあるいは複数のサブディレクトリを削除します。
- RNDIR [/H][/P] 複合ファイルスペック ファイル名
  ひとつあるいは複数のサブディレクトリの名前を変更します。
- SET [名前][セパレータ][値]
  環境変数を表示・設定します。
- TIME [時間]
  現在の時間を表示・設定します。
- TYPE [/H][/P][/A][/B] 複合ファイルスペック | デバイス
  ファイルあるいはデバイスからデータを表示します。
- UNDEL [ファイルスペック]
  直前に削除されたファイルを復活します。
- VER
  システムのバージョン番号を表示します。
- VERIFY [ON | OFF]
  現在のディスク書込みベリファイの状態を表示・設定します。

- VOL [d:][ボリューム名]
  ディスクのボリューム名を表示・変更します。
- XCOPY [ファイルスペック [ファイルスペック]] [/A][/E][/H][/M][/P][/S][/T][/V][/W]
  ひとつのディスクから別のディスクへファイルおよびディレクトリをコピーします。
- XDIR [ファイルスペック][/H]
  ディレクトリ中のすべてのファイルのリストを表示します。

5.3 コマンドの説明

ASSIGN

内部コマンド

機能　論理ドライブを物理ドライブに割り当てます。

書式　ASSIGN [d: [d:]]

解説　パラメータとして ドライブ名が指定されていないと、現在設定されているドライブの割当てがキャンセルされます。

ドライブ名がひとつだけ指定されると、その論理ドライブに割り当てられている物理ドライブ名が表示されます。

どちらのドライブも指定されると、最初のドライブ名で指定されるドライブ（論理ドライブ）に対するMSX-DOSのアクセスが、2番目のドライブ名で指定される物理ドライブに対して行われるようになります。

文例

```
A>ASSIGN
```

すべてのドライブの割当てをキャンセルします。

```
A>ASSIGN A: B:
```

ドライブAをBに割り当てます。したがって、すべてのドライブAへのアクセスは、今後ドライブBへのアクセスとなります。

```
A>ASSIGN A:
A:=B:
```

論理ドライブ名Aに現在割り当てられている物理ドライブ名（この例ではB:）を表示します。

ATDIR

内部コマンド

機　能　ディレクトリの属性を変更し、それらを可視または不可視にします。

書　式　ATDIR +H | -H [/H][/P] 複合ファイルスペック

解　説　複合ファイルスペックで指定したディレクトリの属性を変更します。

ATDIR コマンドでは以下のスイッチが使えます。

/H　不可視を意味し、不可視属性のディレクトリも処理の対象にします。

/P　ページモードを意味し、ディスプレイいっぱいに表示されたところで表示を中断します。表示を再開するには任意のキーを押します。

+Hが指定されると、選択されたディレクトリは不可視状態となり、/Hスイッチが指定されない限り、他のディレクトリ操作コマンドによって影響を受けたり、DIR コマンドで表示したりできないようになります。

Hオプションは指定のディレクトリの属性を可視に変更します。-Hオプションは、/Hスイッチが指定されていない場合には何の効果も持ちません。

このコマンドは、指定のディレクトリ中のファイルやディレクトリの属性を変更しません。

またディレクトリはファイルと異なり、読み出し専用にすることはできません。

エラーが起こると、エラーとなったディレクトリの名前に続いてエラーメッセージが表示され、コマンドは続行します。

ディレクトリの現在の属性は、DIR /H コマンドを使用して表示させることが出来ます。

文　例

```
A>ATDIR +H DIR1
```

DIR1 というディレクトリを不可視状態にします。

```
A>ATDIR -H DIR1/H
```

不可視のディレクトリ DIR1 を可視状態にします。

```
A>ATDIR +H DIR?
```

DIR?にマッチするすべてのディレクトリ（例えば、DIR1、DIR2、DIR3 など）を不可視状態にします。

```
A>ATDIR +H ¥DIR1+¥DIR2
```

DIR1 および DIR2 という2つのディレクトリを不可視状態にします。

ATTRIB

内部コマンド

機能

ファイルの属性を変更し、それらを可視・不可視に、あるいは読み出し専用・書き込み可能にします。

書式

ATTRIB +H | -H | +R | R [/H][/P] 複合ファイルスペック

解説

複合ファイルスペックで指定したファイルの属性を変更します。

ATTRIB コマンドでは以下のスイッチが使えます。

/H 不可視を意味し、不可視属性のファイルも処理の対象にします。

/P ページモードを意味し、ディスプレイいっぱいに表示されたところで表示を中断します。表示を再開するには任意のキーを押します。

「複合ファイルスペック」では、属性を変更するファイルを指定します。/H スイッチが指定されると、不可視のファイルもまたその属性が変更されます。

+H を指定すると、選択されたファイルの属性は不可視に変更されて、/H スイッチが指定されないかぎりほとんどのコマンドによって影響を受けたり、DIR コマンドによって表示されたりしなくなります。

H は選択されたファイルを可視状態にします。-H オプションは、/H スイッチが指定されていない場合には何の効果も持ちません。

+R が指定されると、指定されたファイルは読み出し専用となります。-R は指定されたファイルを読み出し・書き込み可能のファイルとします。読み出し専用ファイルは、書き込みや変更ができません。

エラーが起きると、ファイル名に続いてエラーメッセージが表示されコマンドは続行します。

DIR コマンドを使用して、ファイルの属性を表示することができます。

文例

```
A>ATTRIB +R FILE1
```

FILE1 というファイルを読み出し専用にし、これ以後、変更や削除ができないようにします。

```
A>ATTRIB +H B:¥DIR1¥*.COM
```

B:DIR1 というディレクトリ中のすべての*.COM ファイルを不可視状態にし、DIR コマンドによって表示されないようにします。

```
A>ATTRIB -R -H ¥DIR1/H/P
```

DIR1 中のすべてのファイルを読み出し・書き込み可能として、さらに可視状態にします。画面出力がある場合には1画面ごとに停止します。

```
A>ATTRIB +R ¥DIR1 + ¥DIR2 + FILE1
```

DIR1 および DIR2 というディレクトリ中のすべてのファイル、および FILE1 というファイルを読み出し専用とします。

BASIC

内部コマンド

機　能　MSX Disk BASIC に制御を移します。

書　式　BASIC [プログラム名]

解　説　[プログラム名] は、ディスク上の BASIC のプログラムの名前です。

制御は内蔵の MSX-BASIC に渡され、プログラム名が指定された場合には、その BASIC プログラムがロード後、実行されます。RAM ディスクが設定されている場合には、BASIC でも引続き使用できます。

BASIC のコマンド CALL SYSTEM ("コマンド名") を使用して MSX DOS に戻ることができますが、この場合、MSX DOS で実行可能な任意のコマンドを指定して、それを実行させることができます。コマンドが指定されないと、REBOOT.BAT というバッチファイルを検索し、あれば REBOOT.BAT を実行します (バッチファイルについては、7章を参照)。

文　例

```
A>BASIC
```

MSX-Disk BASIC のモードにします。

```
A>BASIC MYPROG.BAS
```

MSX-Disk BASIC のモードとし、MYPROG.BAS という BASIC のプログラムをロード後、実行します。

BUFFERS

内部コマンド

機能

システムが使用するディスクバッファの数を表示・変更します。

書式

BUFFERS [数値]

解説

数値が指定されないと、システムが現在使用しているディスクバッファの数が表示されます。数値が指定されると、バッファの数が指定の数に変更され、指定した数値が以前のものよりも小さい場合には、不要となったメモリが他の目的のために解放されます。指定されたバッファ数を確保するだけの十分なメモリがない場合には、取れる限りのバッファが取られ、エラーにはなりません。

ディスクバッファの数を増加させると、ある種のアプリケーション、特にファイルへのランダムアクセスを実行するようなものでは、実行速度が向上する可能性があります。ただし、数を10以上に設定しても実行速度はそれほど改善できず、メモリの浪費となってしまいます。

ディスクバッファとして使用されるメモリ領域は環境変数やファイルのオープンにも利用されます。したがって、バッファを可能な限り多く設定したままでは、ある種のコマンド、特にSET、COPYおよびCONCATなどのコマンドが実行できなくなることがあります。これらのコマンドのどれかが「notenough memory」「メモリーが足りません」エラーを出力した場合には、バッファの数を減らすことで対応できる場合があります。ただし、バッファの数を3より小さくすると、実行速度は著しく低下します。

システムのデフォルトのバッファ数は5で、大部分の用途にはこれで十分です。

文例

```
A>BUFFERS
BUFFERS=5
```

ディスクバッファの現在の数を表示します（この例では5）。

```
A>BUFFERS 10
```

バッファの数を、10を最大値にしてできるかぎり多く増やします。

```
A>BUFFERS=5
```

バッファの数を再び5に設定します。

CD

内部コマンド

機　能	CHDIR コマンドと同じです。CHDIRコマンド（P.79）を参照して下さい。

CHDIR

内部コマンド

機能　カレントディレクトリを表示・変更します。

書式　CHDIR [d:][パス]

または

CD [d:][パス]

解説　パスが指定されない場合には、カレントドライブ、あるいは指定ドライブのカレントディレクトリパスが表示されます。これは、ルートディレクトリからカレントディレクトリへのディレクトリパスです。

パスが指定された場合には、カレントドライブ、あるいは指定ドライブのカレントディレクトリが、パスによって指定されたディレクトリに変更されます。

どのドライブも固有のカレントディレクトリを持っています。カレントディレクトリはそのドライブについて、最後にCHDIRコマンドによって指定されたディレクトリ（最初にいるディレクトリはルートディレクトリ）になっています。そして、新たにCHDIRコマンドを指定するか、あるいはディレクトリがアクセスされたときにそのディレクトリが見つからない場合（例えばディスクが交換されたような場合）まで有効となります。後者の場合には、カレントディレクトリはルートディレクトリとなります。

CDコマンドはCHDIRコマンドの短縮形で、簡便さのためと MS DOSとの互換性のために提供されています。

コマンドプロンプトは、SET PROMPT ONコマンドを使用してカレントディレクトリを表示するように変更できます（8章の環境変数についての記述を参照）。

文例

```
A>CHDIR ¥DIR1
```

カレントドライブのカレントディレクトリを DIR1 に変更します。

```
A>CHDIR A:DIR2
```

ドライブ A のカレントディレクトリを、そのサブディレクトリの DIR2 に変更します。

```
E>CD
E:¥DIR1
```

カレントドライブのカレントディレクトリを表示します（この例ではDIR1）。

```
A>CHDIR A:
A:¥DIR2
```

ドライブAのカレントディレクトリを表示します（この例ではDIR2）。

CHKDSK

外部コマント

機　能	ファイルシステムの整合性を検査します。
書　式	CHKDSK [d:][/F]

解　説

指定の、あるいはカレントドライブのファイルシステムのデータ構造の整合性を調べ、失われたディスクスペースをチェックします。ディスクにエラーが発見されると、修復が行われます。クラスタが失われていると、失われたディスクスペースを使用可能なディスクスペースに変換するか、あるいはファイルに変換するかについて、入力を要求します。後者が選択されると FILE0000.CHK、FILE0001.CHK などの形式のファイルがルートディレクトリに作成されます。

CHKDSK コマンドでは以下のスイッチが使えます。

/F　実際にディスクの修復を行います。

/F スイッチが指定されていないと、CHKDSK は実際にはディスクの修復を行いませんが、修復を行ったときと同じメッセージを表示します。このメッセージによって、/F が指定されたときにディスクにどのような処理が行われるのかをあらかじめ確認できます。

ある種のプログラムが中断されたときに、ディスクスペースが失われる (「失われたクラスタ」が発生する) ことがあります。これは特に CP/M のプログラムについてあてはまります。

CHKDSK はすべてのディスクについて定期的に実行するとよいでしょう。

メディアタイプがMSX-DOS2 フォーマットでない場合、その旨を表示する機能が ve[illegible]した。

```
A>CHKDSK B:

Insert disk for drive B:
and strike a key when ready

Disk is not formatted by MSX-DOS2

    713K total disk space
    620K in 89 user files
     93K available disk space
```

文　例

```
A>CHKDSK
```

カレントドライブのディスクが検査され、「ステータスレポート」が出力されます。エラーが見つかった場合も、ディスクは修復されません。

```
A>CHKDSK B:
```

ドライブBのディスクが検査されます。エラーが見つかった場合も、ディスクは修復されません。

```
A>CHKDSK /F
20 lost clusters found in 1 chain
Convert lost chains to files (Y/N)?
```

```
A>CHKDSK /F
20個の失われたクラスタが1個のチェイン中に見つかりました.
失われたチェインをファイルに変換しますか (Y/N)?
```

カレントドライブのディスクが検査され、破損クラスタが検出されました。/F が指定されているため、ディスクが修復され、破損箇所が回復します。

CLS

内部コマンド

機　能　　画面をクリアします。

書　式　　CLS

解　説　　画面をクリアして、カーソルをホームポジションに移動します。

文　例

```
A>CLS
```

画面がクリアされます。

COMMAND2

外部コマンド

機　能　　コマンドインタープリタを起動します。

書　式　　COMMAND2 [コマンド]

解　説

「コマンド」は通常プロンプトで入力できる任意のコマンド（例えば本書に記載されているコマンド）です。

COMMAND2はディスク上のコマンドインタープリタの名前であり、外部コマンドとして実行することができます。これは通常、システムの起動時にMSXDOS2.SYSによってロード、実行されます。それによって本書中のすべてのコマンドを実行することができるようになります。

しかし、場合によっては種々の理由でもう1度コマンドインタープリタを起動したい場合があります。例えば、2番目に起動されたCOMMAND2.COMはより新しいバージョンで、より多くの機能を提供しているような場合が考えられます。ある種の複雑なプログラムのように、外部プログラムが他のプログラムをロード、実行できる場合、COMMAND2.COMをロードして任意のMSX DOSコマンドを指定することができます。EXITコマンドでCOMMAND2.COMを終了すると、制御は呼び出したプログラムに戻ります。

パラメータとしてコマンドが指定されないと、COMMAND2.COMは単に通常のプロンプトを出力し（AUTOEXEC.BATあるいはREBOOT.BATを実行せずに）、通常のコマンド待ちの状態となります。EXITコマンドが入力されると、起動されたCOMMAND2.COMは終了し、元のCOMMAND2.COMまたはプログラムに戻ります（P.103「EXIT」を参照）。このEXITコマンドにエラーコードが与えられると、元のコマンドインタープリタまたはプログラムがそれを受け取り、COMMAND2.COMとMSXDOS2.SYSの場合には、適切なエラーメッセージが出力されます（9章「エラーおよびメッセージ」を参照）。

一方、COMMAND2.COMへのパラメータとしてコマンドが指定されると、そのコマンドが通常の方法で入力されたのと同様に実行されます。コマンドは内部コマンドでも、外部コマンドのCOMあるいはBATファイルでもかまいません。コマンドを実行し終ると、COMMAND2.COMはそのまま元のコマンドインタープリタ、あるいはプログラムに制御を戻します。

このような方法で、通常のコマンドインタープリタから2番目のCOMMAND2.COMを、バッチファイル名をコマンドに指定して起動すると

バッチファイルを「チェイン」する代わりに「ネスト」することができます（バッチファイルについては、7章を参照）。

COMMAND2.COM が実行されると、すべての環境変数がセーブされ、終了するときは元に戻されます。このようにして、新たに起動されたCOMMAND2.COM は最初の環境変数を引き継ぎます。ただし、環境変数が未定義である場合には、デフォルトの環境変数を設定します。新たに起動されたCOMMAND2.COM が実行されている間に行われた環境変数の変更は、そのCOMMAND2.COM が実行されている間だけ有効であり、終了すると失われます。

C●MMAND2.COMが起動されるごとにメモリが消費されますが、終了するとそのメモリは再び解放されます。これは、環境変数の数に依存しますが、通常は1.5Kバイト程度です。

COMMAND2.COMが外部プログラムを実行すると、プログラムはCOMMAND2.COM が占有していたメモリの一部を使用することがあります。その場合、COMMAND2.C●M はプログラム終了後に自分自身をディスクから再ロードしなければなりません。その時、COMMAND2.C●M はファイルの場所を検索するために、環境変数SHELL を利用します（環境変数については8章を参照）。ディスク上のCOMMAND2.COM が最初にロードされたとき SHELL はそのファイルを参照するように設定されます。

文 例

```
A>COMMAND2
A>
```

新たにCOMMAND2.COM がロードされ、通常のプロンプトを出力します。EXIT によって元のCOMMAND2.C●M に戻ります。

```
A>COMMAND2 FILE.BAT
```

通常、バッチファイル中で指定します。FILE.BAT というファイルが実行され、それが終了するとこのコマンドの次のコマンドから現在のバッチファイルが再開されます。

CONCAT

内部コマンド

機　能　ファイルを連結します（つないでひとつにします）。

書　式　CONCAT [/H][/P][/B][/V][/A] 複合ファイルスペック ファイルスペック

解　説　複合ファイルスペックでは連結するファイルを指定します。

2番目のパラメータのファイルスペックはワイルドカードを含んではならず、そのファイルはソースファイルが読まれる前に生成されます。その後それぞれのファイルが読まれ、直前のファイルの終りに連結されて、目的のファイルに書き出されます。

それぞれのソースファイルが読まれるごとに、そのファイル名が出力されます。何らかの理由でそのファイルが読めない場合（例えば、それが出力ファイルとして生成されているような場合）には、そのファイル名の後にエラーメッセージが続き、CONCATの処理は次のソースファイルから続けられます。

CONCATコマンドでは以下のスイッチが使えます。

/H　不可視ファイルを連結可能とします。

/P　出力を1画面ごとに（何らかのキーが押されるまで）停止します。多くのファイルを連結させるような場合に使います。

/B　バイナリファイルとして扱います。
読み込まれるデータはそのまま扱われ、何のデータも付け加えられません。また、/Bは出力ファイルや複合ファイルスペック中の任意のファイルスペックについて、指定することができます。その場合には、/Bはそれらのファイルについてのみ有効となります。

/A　ASCIIファイルとして扱います（デフォルトです）。

/V　CONCATコマンドの実行中、書き込みチェックが有効になります（P.146「VERIFY」を参照）。
これによってベリファイ機能を持つディスクドライバを使用していれば、データがディスクに正しく書き込まれることが保証されますが、書き込みチェックをするぶん処理時間がかかります。

通常、連結はASCIIファイルに対して行われます。それぞれのソースファイルは、最初にエンドオブファイルを表す文字（CTRL＋Z）が現れるまで読み込まれ、すべてのデータが書き出された後、最後にエンドオブファイル文字を出力して終了します。

CONCATが「Not enough memory」「メモリー不足です」エラーを出力する場合には、バッファ数を減少させる（P.76「BUFFERS」を参照）かいくつかの環境変数を除去（8章の環境変数についての記述を参照）して十分なメモリを確保して下さい。

```
A>CONCAT *.DOC ALL.PRN
```

ALL.PRNという新しいファイルが生成され、*.DOCに適合するすべてのファイル（例えば、FILE1.DOC、FILE2.DOC、FILE3.DOCなど）が連結されて、ディスク上で見つかった順にそれらが新しいファイルに書き出されます。もしALL.PRNというファイルがすでに存在している場合はそこに重ね書きされます。

```
A>CONCAT /H /P *.DOC ALL.DOC
FILE1.DOC
FILE2.DOC
FILE3.DOC
ALL.DOC -- Destination file cannot be concatenated
```

```
A>CONCAT /H /P *.DOC ALL.DOC
FILE1.DOC
FILE2.DOC
FILE3.DOC
ALL.DOC -- 複写先ファイルは結合できません
```

ALL.DOCという新しいファイルが生成され、*.DOCに適合するすべてのファイルが連結されて、ディスク上で見つかった順に新しいファイルに書き出されます。出力ファイルのALL.DOCもソースファイル名*.DOCに適合するため、メッセージが出力され、それは連結されるファイルには含まれません。また/Hが指定されているため、不可視ファイルも結合され、/Pが指定されているので、表示が1画面を越えるときは、1画面ごとに停止しキー入力を待ちます。

```
A>CONCAT /B FILE2.DOC+FILE3.DOC+FILE1.DOC ALL.DOC
```

ALL.DOCという新しいファイルが生成され、FILE2.DOC、FILE3.DOCおよびFILE1.DOCがその順番で連結されて、そのファイルに書き出されます。これらのファイルはバイナリモードで連結されます。

COPY

内部コマンド

機　能	ファイルまたはデバイスから、他のファイルまたはデバイスへデータをコピーします。
書　式	COPY [/A][/H][/T][/V][/P][/B] 複合ファイルスペックファイルスペック
解　説	複合ファイルスペックでは、コピー元のファイルを指定します。デバイスの指定が含まれていてもかまいません。

ファイルスペックでは、コピー先のファイルを指定します。コピー先のファイルの定義を次に示します。

- [d:][パス][ファイル名] | デバイス

  ここでd:とパスは、デフォルトではそれぞれカレントドライブ、カレントディレクトリとなります。ファイル名の一部分にワイルドカードを含む場合にはソースファイル名から適切な文字が代入されます。ファイル名が与えられないと、ソースファイルの名前がそのまま使用されます。コピー先のファイルにディレクトリを指定すると、ファイル名を*.*として、ファイルがそのディレクトリにコピーされます。

COPYはファイルを書き出す前にできる限り多くのソースファイルをメモリ中に読み込み、メモリがいっぱいになった時点で、読み込まれた順にファイルを書き出します。それぞれの出力ファイルが生成されるたびに、ソースファイル名が出力されます。出力ファイルが作成不可能な場合には、エラーメッセージが出力され、コピー処理は次のファイルに進みます。

読み出し専用ファイルが同一の名前ですでに存在している場合などCOPYが出力ファイルを作成できない原因は多く存在します。ユーザーが誤りを犯しているかもしれないような場合には、COPYは出力ファイルを作成しません。例えば、ファイルを自分自身にコピーすることはできず、また複数のファイルをひとつのファイルへコピーすることもできません。ひとつのファイルの出力が以前のソースファイル、あるいはすでに他に使用されているファイル（例えば現在実行中のバッチファイル）の内容を消してしまうような場合には、「Cannot create destination」「ファイルを作成できません」エラーが出力されます。また、多くのファイルをひとつのファイルにコピーしようとすると、「Cannot overwrite previousdestination file」「ファイルの重ね書きができません」エラーとなります。これは通常、コピー先のファイルにディレクトリを指定しようとして、その名前を間違って指定した場合に起こります。ただし、コピー先のファイルがデバイスの場合は、エラーにはなりません。

COPY コマンドでは以下のスイッチが使えます。

/H　不可視属性のファイルもコピーします。

/P　1画面ごとにメッセージ出力を停止させます。

/A　ASCII コピーが実行されます。ソースファイルの最初のエンドオブファイル（EOF）文字（CTRL ＋ Z）までを読み込み、それぞれの出力ファイルには、ファイルの最後に EOF 文字が付け加えられます。また、/A は出力、あるいは複合ファイルスペック中の任意のファイルスペックに個別に指定することが可能で、その場合には、指定したソース、あるいは出力にのみ有効となります。

/B　バイナリコピーを行います。読み込まれるファイルがそのままコピーされ、ファイルには何もデータが付け加えられません。

/V　COPY コマンドの処理の間、ディスクドライバがベリファイ機能を持っていれば書き込みチェックを行います（P.146「VERIFY」を参照）。これによってデータがディスクに正しく書き込まれることを保証しますが、処理にかかる時間は増加します。

/T　出力ファイルには現在の日付と時間が設定されます。

出力ファイルはソースファイルの属性にかかわらず可視であり、読み書き可能として作成されます。ATTRIB コマンドで、これらの属性を変更することができます。

/T スイッチが指定されなかった場合、出力ファイルにはソースファイルと同じ日付と時間が設定されます。

COPY によって「Not enough memory」「メモリー不足です」エラーが起こるときには、バッファ数を減少させる（P.76「BUFFERS」を参照）か、あるいはいくつかの環境変数を除去（8章「環境変数の設定」を参照）して十分なワークエリアを確保して下さい。

COPY コマンドはファイルの連結（ファイルをつなげる）をサポートしていないため、MS-DOS や MSX-DOS1 のものよりも機能が簡潔になっています。ファイルの連結を行うには、CONCAT コマンドを使用して下さい（P.86「CONCAT」を参照）。

```
A>COPY FILE1 B:
```

FILE1 というファイル名のファイルを、カレントドライブのカレントディレクトリから、ドライブ B のカレントディレクトリへ同じファイル名でコピーします。

```
A>COPY /H MSXDOS2.SYS + COMMAND2.COM B:
```

/H スイッチで MSXDOS2.SYS および COMMAND2.COM という2つの不可視ファイルをドライブBにコピーします。ブートディスクを作成するような場合に使用します。

```
A>COPY A:¥DIR1 B:¥DIR1 /V
```

ドライブAのルートにあるディレクトリ DIR1中のすべてのファイルをドライブBの同名のディレクトリにコピーします。その際に、ファイルが正しく書き込まれているか書き込みチェックを行います。

```
A>COPY B:
```

ドライブBのカレントディレクトリ中のすべてのファイルをカレントドライブのカレントディレクトリにコピーします。

```
A>COPY A:*.DOC B:/T
```

.DOC に適合するすべてのファイル（例えばFILE1.DOC、FILE2.DOC、FILE3.DOC など）をドライブBのカレントディレクトリにコピーし、.DOC ファイルの日付および時間の代わりに、現在の日付と時間をコピーされたファイルに設定します。

```
A>COPY *.BAT
AUTOEXEC.BAT -- File cannot be copied onto itself
REBOOT.BAT -- File cannot be copied onto itself
      0 files copied
```

```
A>COPY *.BAT
AUTOEXEC.BAT -- 自分自身にはコピーできません
REBOOT.BAT -- 自分自身にはコピーできません
      0個のファイルをコピーしました
```

この例では、*.BATに適合するすべてのファイル（ここではAUTOEXEC.BAT と REBOOT.BAT）をカレントドライブのカレントディレクトリ内でコピー

するように COPY を使用しているが、COPY はメッセージを出力してこれを警告しています。この場合には、ファイルは実際にはコピーされていません。

```
A>COPY *.BAT DIR2
AUTOEXEC.BAT
DIR2 -- Cannot overwrite previous destination file
      1 file copied
```

```
A>COPY *.BAT DIR2
AUTOEXEC.BAT
DIR2 -- ファイルの重ね書きができません
      1 個のファイルをコピーしました
```

この例では、*.BATに適合するすべてのファイル（ここではAUTOEXECBATとREBOOT.BAT）をDIR2というディレクトリにコピーするためにCOPYを使用しています。しかし、DIR2は存在していなかったためDIR2はファイル名として解釈されました。したがってAUTOEXEC.BATをDIR2というファイルにコピーし、それからREBOOT.BATもDIR2というファイルにコピーしようとしました。これはたぶん間違いであろう（この場合DIR2というディレクトリが存在しない）ということで、警告のメッセージが表示されました。REBOOT.BATは実際にはどこにもコピーされていません。

DATE

内部コマンド

機　能	現在の日付を表示・設定します。

書　式	DATE [日付]

解　説

このコマンドの後に日付を指定するとその日付に設定されます（入力形式については下記参照）。コマンドの後に日付が指定されないと現在の曜日と日付が出力され、新しい日付の入力待ちとなります。ここで何も入力しないと（つまり、[↵] キーだけを押すと）、現在の日付は変更されません。入力があるとその入力は新しい日付であると見なされ、後述のように解釈されます。日付が無効であるとエラーメッセージが表示されて再び新しい日付の入力待ちとなります。

日付は3組までの数字で構成され、それぞれは次のような区切り文字で区切らなければなりません。

空白　タブ　　　. / :

文字のどちらかの側に空白があってもかまいません。数字の入力を省略したフィールドには現在の設定が使用されます。年の入力は4桁でも下2桁でもかまいません。後者の場合には、上2桁は年が80以上のときには19、小さいときには20として解釈されます。

日付の表示および入力の形式には柔軟性があり、変更することができます。DATEという環境変数がそのMSXマシンが使用される国の形式に適合するように、デフォルトで設定されています（8章の環境変数についての記述を参照）。

例えば、日本向けのマシンではデフォルトの設定はYY MM DDです。

SET DATE DD-MM-YY というコマンドで、日付の形式をヨーロッパ形式に変更することができます。この形式はDIRコマンドによって表示される日付にも反映されます。

環境変数DATEが定義されていると、それはDATEコマンドによって日付の入力で必要な形式として示されます。

文　例

```
A>DATE 91-9-20
```

現在の日付を1991年9月20日に設定します。

```
A>DATE
Current date is Sat 1991-09-20
Enter new date (yy-mm-dd): --21
```

```
A>DATE
現在の日付は (土) 1991-09-21 です
新しい日付を入力して下さい (yy-mm-dd): --21
```

パラメータが入力されなかったので現在の日付 1991 年 9 月 20 日が表示され、新しい日付の入力待ちとなります。プロンプトへの応答において 21 日を指定するだけで、日付は翌日に更新されました。年と月は指定を省略したため、変更されません。

```
A>SET DATE = DD/MM/YY
```

日付の形式をヨーロッパ形式に変更します。

```
A>DATE
Current date is Sun 20-09-1991
Enter new date (DD/MM/YY):
```

```
A>DATE
現在の日付は (日) 20-09-1991 です
新しい日付を入力して下さい (DD/MM/YY):
```

パラメータが入力されなかったので現在の日付 1991 年 9 月 21 日がヨーロッパ形式で表示され、プロンプトが出力されます。プロンプトに対する応答はヨーロッパ形式で行います。

その他の形式は次の通りです。

YY/MM/DD	ISO
MM/DD/YY	American
DD/MM/YY	European

DEL

内部コマンド

機　能　ひとつあるいは複数のファイルを削除します。

書　式　DEL [/H][/P] 複合ファイルスペック

または

ERA [/H][/P] 複合ファイルスペック

または

ERASE [/H][/P] 複合ファイルスペック

解　説　複合ファイルスペックでは削除するファイルを指定します。

DEL コマンドでは以下のスイッチが使えます。

/H　不可視ファイルも削除できます。

/P　出力を1画面ごとに停止させることができます。

削除の処理中、ファイルが何らかの理由で（例えば「読み出し専用」にセットされている場合）削除できないと、そのファイル名がエラーメッセージとともに出力され、削除の処理は次のファイルへ進みます。

ファイル名が*.*である場合

```
Erase all files (Y/N)?
```

```
全てのファイルを消去しますか (Y/N)?
```

というプロンプトが表示され、ユーザーの応答待ちとなります。応答が「y」あるいは「Y」以外である場合には、ファイルの削除は行われません。これは、ディレクトリ中のすべてのファイルを誤って消去してしまわないための配慮です。

MSX-DOS2 でフォーマットされたディスク上で削除されたファイルは、削除の直後に UNDEL コマンドを使用すると復活することができます。

文　例

```
A>DEL FILE1.BAK
```

FILE1.BAK というファイルをカレントドライブのカレントディレクトリから削除します。

```
A>DEL *.COM/H
```

*.COM に適合するすべてのファイルを可視・不可視にかかわらず削除します。

```
A>DEL B:¥UTIL¥*.COM+B:¥UTIL¥*.BAT
```

.COM または.BATに適合するすべてのファイルを、ドライブ B の UTIL というディレクトリから削除します。

```
A>DEL B:¥UTIL
Erase all files (Y/N)?
```

```
A>DEL B:¥UTIL
全てのファイルを消去しますか (Y/N)?
```

ドライブ B の UTIL というディレクトリ中のすべてのファイルを削除します。多くのファイルが削除されるため、プロンプトが最初に表示され誤消去を防ぎます。

```
A>DEL *.BAT
AUTOEXEC.BAT -- Read only file
REBOOT.BAT -- Read only file
```

```
A>DEL *.BAT
AUTOEXEC.BAT -- ファイルが読み出し専用です
REBOOT.BAT -- ファイルが読み出し専用です
```

属性が読み出し専用となっていた AUTOEXEC.BAT および REBOOT.BAT を除いて、*.BAT に適合するすべてのファイルを削除します。

DIR

内部コマンド

機能	ディスク上のファイル名を表示します。
書式	DIR [/H][/W][/P][複合ファイルスペック]
解説	「複合ファイルスペック」は、表示すべきファイルを指定します。

DIR コマンドでは以下のスイッチが使えます。

/H　不可視ファイルも表示されます。

/W　ワイド形式で表示され、1行に複数のファイル名が出力されます。

/P　出力は1画面ごとに停止し、キー入力待ちとなります。

DIR コマンドでは他のすべてのコマンドとは異なり、主ファイル名やファイル名拡張子を指定しなくてもよく、どちらも省略時には「*」と解釈されます。したがって、ファイル名「FRED」は「FRED.*」と同等で、ファイル名「.COM」は「*.COM」と同等です。主ファイル名の最後に「.」が指定されると、拡張子も指定されているものと見なされ、ファイル名「FRED.」は上記の例とは異なり「FRED.*」とは同等とは見なされません。

表示には2つの形式があります。/W スイッチを指定すると、リストはワイド形式で表示され、1行に複数のファイル名が出力されます。サブディレクトリ名、ファイルの属性、それぞれのファイルが作成された日付と時間は、表示されません。

/W スイッチを指定しないと、それぞれのファイル名はその属性、ファイルサイズ、最後に作成された日付と時間とともに1行に1ファイル表示されます。属性はファイルが読み出し専用の場合に「r」、不可視ファイルである場合（/H が指定されている場合）に「h」として表示されます。ファイルの時間がゼロ（ファイルがそれに関連した時間情報を持たない）の場合、時間は表示されません。ファイルの日付がゼロだと、日付も時間も表示されません。日付と時間の表示形式は変更することができます（P.92「DATE」および P.139「TIME」を参照)。

/W を指定しない表示は 40 文字画面に収まるようにデザインされていますが、それよりもディスプレイの桁数が少ない場合には、表示を1行に収めるためにリストの一部の項目が表示されません。/W が指定されたときに表示される1行ごとのファイル数も画面の幅にしたがって調整されます。しかし、表示幅が13文字よりも小さい場合には、どちらの場合もファイル名は次の行にまたがって表示されます。

ファイルのリストの一番始めにはディスクのボリューム名と表示されるディレクトリ名が出力されます。一番終りには表示されたファイルの数、ファイルの合計バイト数、未使用のディスク領域の合計（バイト数表示は、1K以上の場合にはキロバイト単位で表示され、端数は切り捨てられます）が表示されます。

サブディレクトリのディレクトリが表示されるとき、リストされる最初の2つの項目は常に「.」、「..」と呼ばれる特殊なディレクトリです。これらは新しいディレクトリが作成されるときに自動的に作成され、これにより、パス名中で「.」、「..」によって、それぞれカレントディレクトリ、親ディレクトリを指定できます（パスの記述については、5.1の「この章の表記法」を参照）。

文　例

```
A>DIR
 Volume in drive A: is MSX-DOS 2
 Directory of A:¥
 MSXDOS2   SYS  r    4480  90-09-03 4:58p
 COMMAND2  COM  r   15472  90-09-11 4:24p
 UTILS              <dir>  90-09-20 6:40p
 HELP               <dir>  90-09-20 6:40p
 14K in 2 files 222K free
```

```
A>DIR
 ドライブ A:のボリューム名は MSX-DOS 2
 ディレクトリ A:¥
 MSXDOS2   SYS  r    4480  90-09-03 4:58p
 COMMAND2  COM  r   15472  90-09-11 4:24p
 UTILS              <dir>  90-09-20 6:40p
 HELP               <dir>  90-09-20 6:40p
 14Kバイトを2個のファイルで使用中
 222Kバイトが使用可能です
```

カレントドライブのカレントディレクトリ中のすべてのファイル名およびディレクトリ名を表示します。

この例からわかるように、このディスクはMSXDOS2.SYSとCOMMAND2

COMという読み出し専用のMSX DOSシステムファイルと、UTILSとHELPというディレクトリを含んでいます。

```
A>DIR B:¥HELP/W
Volume in drive B: is MSX-DOS 2
Directory of B:¥HELP
BUFFERS .HLP    ATTRIB  .HLP    ASSIGN  .HLP
ATDIR   .HLP    CHDIR   .HLP    CD      .HLP
SYNTAX  .HLP    ENV     .HLP    BATCH   .HLP
EDITING .HLP
25K in 10 files 222K free
```

```
A>DIR B:¥HELP/W
ドライブB:のボリューム名はMSX-DOS 2
ディレクトリ B:¥HELP
BUFFERS .HLP    ATTRIB  .HLP    ASSIGN  .HLP
ATDIR   .HLP    CHDIR   .HLP    CD      .HLP
SYNTAX  .HLP    ENV     .HLP    BATCH   .HLP
EDITING .HLP
25Kバイトを10個のファイルで使用中
222Kバイトが使用可能です
```

ドライブBのディレクトリHELPをワイド形式で表示します。

```
A>DIR UTILS + HELP/P
```

UTILSおよびHELPというディレクトリ中のすべてのファイルを表示し、1画面ごとに表示を停止します。

```
A>DIR .COM
```

主ファイル名が指定されていないため、デフォルトの「*」が使用されます。したがってこのコマンドは、コマンドDIR *.COMと同等です。

```
A>DIR COMMAND2
```

拡張子が指定されていないため、これはデフォルトの「.*」が使用されます。したがってこのコマンドは、コマンドDIR COMMAND2 *と同等です。

DISKCOPY

外部コマンド

機　能　ディスクを別のディスクにコピーします。

書　式　DISKCOPY [d: [d:]] [/X][/S]

解　説　最初のドライブは複写元のドライブ、2番目のドライブは複写先のドライブ（デフォルトはカレントドライブ）です。ドライブを指定しないと、DISKCOPYは複写元、複写先のドライブについて、プロンプトを出します。

DISKCOPY を使用する前に、複写先のディスクは複写元のディスクと同じフォーマットにしておかなければなりません。同一のフォーマットでない場合にはエラーとなります。

DISKCOPY コマンドでは、以下のスイッチが使えます。

/X　ディスクコピー処理の間に出力されるいろいろなメッセージが表示されません。

/S　ブートコードもコピーします（ver.2.30 から追加されたスイッチです）。

COPY *.*と DISKCOPY の相違点は、前者がファイルごとにコピーを行うのに対し、後者はディスクの内容を書かれている通りにそのままコピーするところにあります。

文　例

```
A>DISKCOPY A: B:
Insert source disk in drive A:
Insert target disk in drive B:
Press any key to continue...
```

```
A>DISKCOPY A: B:
複写元ディスクをドライブ A:
複写先ディスクをドライブ B:
何かキーを押して下さい...
```

ドライブA中のディスクをドライブB中のディスクにコピーするコマンドの指定です。したがって、ドライブB中のディスクのすべての既存のデータは破壊されます。始めにプロンプトが表示されます。

```
A>DISKCOPY B:
```

ドライブB中のディスクをカレントドライブ中のディスクにコピーします。

```
A>DISKCOPY
Source drive?
Target drive?
```

```
A>DISKCOPY
複写元ドライブは?
複写先ドライブは?
```

DISKCOPYコマンドがパラメータなしで使用されたため、複写元と複写先のディスクについてプロンプトが表示されています。このプロンプトに対しての応答は、ディスクドライブを表す1文字です。

ECHO

内部コマンド

機　能	指定したテキストを表示します。
書　式	ECHO [テキスト]
解　説	テキストをそのまま画面に表示します。テキストを指定しないと空行が出力されます。 このコマンドは、ECHOという環境変数（8章「環境変数の設定」を参照）によって制御されるバッチファイル中の「echo」の状態とは別ですので混同しないようにして下さい。

文　例

```
A>ECHO AUTOEXEC batch file executed
AUTOEXEC batch file executed
```

指定のテキスト（「AUTOEXEC batch fileexecuted」）が画面に出力されます。

```
A>ECHO
```

パラメータが与えられていないので、空行のみが出力されます。

ERA

内部コマンド

機　能	DEL コマンドと同じです。DEL コマンド（P.94）を参照して下さい。

ERASE

内部コマンド

機　能	DEL コマンドと同じです。DEL コマンド（P.94）を参照して下さい。

EXIT

内部コマンド

機　能　COMMAND2.COM を終了して、呼び出したプログラムに戻ります。

書　式　EXIT [数値]

解　説

数値はエラーコードでデフォルトは 0 です。0 は MSX DOS2 ではエラーなしを示します（エラーについては9章を参照)。

EXIT はコマンドインタープリタ（COMMAND2.COM）を終了してコマンドインタープリタをロード、実行（P.84「COMMAND2」を参照）したプログラム（COMMAND2.COM、他のプログラムあるいは通常 MSXDOS2.SYS）へエラーコードを返します。MSXDOS2.SYS の場合には適当なエラーメッセージが表示され、COMMAND2.COM が再ロード実行されます。

COMMAND2.COM はロード時に現在の環境（8章「環境変数の設定」を参照）をセーブし、EXIT はセーブされた環境を元に戻します。したがって、EXIT によって MSXDOS2SYS に戻る（トップレベルで EXIT を実行する）と環境はクリアされ、COMMAND2COM の再ロード後、デフォルトの環境が再設定されます。これによって、環境をそのデフォルトの値にリセットすることができます。

文　例

```
A>EXIT
```

コマンドインタープリタを終了します。これに続く動作は何がそれをロードしていたかによって異なります。

```
A>EXIT 40
*** User error 40
```

```
A>EXIT 40
*** ユーザーエラー 40
```

コマンドインタープリタをエラーコード 40 で終了します。これはシステムに登録されているエラーではないため、エラーメッセージは最初にコマンドインタープリタをロードしたものによって表示されます。エラーについては9章を参照して下さい。

FIXDISK

外部コマンド

機能	ディスクを完全なMSX DOS2フォーマットに更新します。
書式	FIXDISK [d:] [/S]

解説

d:はFIXDISKで処理するドライブを指定します。指定されなかった場合はカレントドライブに対して処理が行われます。

FIXDISK コマンドでは以下のスイッチが使えます。

/S　ブートセクタを完全にMSX DOS2互換にします。

このコマンドは主に問題のあるMSX DOS1のディスクを完全なMSX DOS2互換のディスクに更新するために使用しますが、同様なフォーマットの他のディスクを更新する場合や、正しくないブートセクタを修正するときにも有用です。

MSX DOS1とMSX DOS2で使用しているディスクの形式は標準に従ったものですが、MSX DOS1ではブートセクタにある情報を参照していないため、MSX DOS1のディスクのブートセクタの情報は必ずしも正しいとは限りません。このようなディスクをMSX-DOS2で使用すると問題が生じます。

FIXDISK コマンドはこのようなディスクのブートセクタの情報を正しく更新し、MSX-DOS2で使用できるようにします。

/Sスイッチが指定されなかった場合、FIXDISKコマンドはMSX DOS2で使用できるような最小限の更新を行います。

このようにして更新されたディスクは、全く異なったフォーマットを採用していない限り、元のシステムとの互換は保たれます。

しかし、MSX-DOS2のUNDELコマンドはMSX-DOS2でフォーマットされたブートセクタに「ボリュームID」(13章「ディスクファイルの構造」を参照)のあるディスクでのみ使用でき、MSX-DOS1や他のシステムでフォーマットされたディスクでは動作しません。

また、間違ったディスクが挿入されていてもそれを検出できません。/Sスイッチは、ブートセクタをMSX DOS2用に書き換え、MSX DOS2用のディスク機能を完全に活用できるようにします。

しかしこのようにして更新されたディスクは元のシステムとの完全な互換性は保たれなくなります。例えば標準外のブートプログラムを使用して

いる一部のゲームなどのアプリケーションディスクに対して、/S スイッチを与えるとそのディスクは MSX-DOS1 や MSX-DOS2 を立ち上げることはできますがアプリケーションを立ち上げることができなくなります。

他のシステムのブートディスクを間違って更新してしまうことのないように、ディスクの更新の前にプロンプトが出力されます。

文 例

```
A>FIXDISK B: /S
Disk in drive B: will only be able to boot MSX-DOS
Press any key to continue
```

```
A>FIXDISK B: /S
ドライブB: のディスクは MSX-DOS しか立ち上げることが出来な
くなります
何かキーを押して下さい
```

ドライブ B のディスクが MSX DOS2の完全互換のものに更新されます。ディスクが他のシステムのブートディスクかも知れないので、ディスクが実際に更新される前にプロンプトが出力されます。

FORMAT

内部コマンド

機能

ディスクをフォーマット（初期化）します。

書式

FORMAT [d:]

解説

指定のあるいはカレントドライブがフォーマットされ、ディスク上のすべてのデータは破棄されます。

FORMAT コマンドを入力した後、マシンによってはオプションがプロンプトで表示され、フォーマットの種類（1DD、2DD など）を選択することができます。これらのプロンプトの内容はMSXマシンのメーカーによって異なりますので、実際にフォーマットを行なうときは、本体の取り扱い説明書にしたがって下さい。

フォーマットが終るとディスク上にはファイルもディレクトリも存在せず、ディスクの全領域が解放されます。その時、ディスクにはボリューム名がついていませんが、VOL コマンド（P.147参照）によって指定することができます。MSX-DOS が起動できるようにディスクをブートディスクにするには、COPY コマンド（P.88参照）を使用して MSXDOS2.SYS と COMMAND2.COM ファイルをコピーしなければなりません。

文例

```
A>FORMAT A:
1 - Single sided
2 - Double sided

? 2

All data on drive A: will be destroyed
Press any key to continue
```

```
A>FORMAT A:
1 - Single sided
2 - Double sided

? 2

ドライブ A:上の全てのデータは消去されます
何かキーを押して下さい
```

ドライブ A のディスクをフォーマットするコマンドを実行しました。この場合、片面と両面のディスクを選択するオプションが利用できるので、両面を選択しました。それから標準の警告プロンプトが出力されます。

```
A>FORMAT
1 - Single sided
2 - Double sided

? 2

All data on drive A: will be destroyed
Press any key to continue...
```

```
A>FORMAT
1 - Single sided
2 - Double sided

? 2

ドライブ A:上の全てのデータは消去されます
何かキーを押して下さい.
```

(ver.2.20 のメッセージ)

ver.2.20 では、フォーマットするドライブを指定されなかったときは、カレントドライブをフォーマットします。

```
A>FORMAT
Drive name? (A,B) A

1 - Single sided
2 - Double sided

? 2

All data on drive A: will be destroyed
Press any key to continue
```

```
A>FORMAT
ドライブ名は? (A,B) A

1 - Single sided
2 - Double sided

? 2

ドライブ A:上の全てのデータは消去されます
何かキーを押して下さい
```

(ver.2.30 以降のメッセージ)

ver.2.30以降では、フォーマットするドライブを指定されなかったときは、フォーマットできるドライブを表示するので、その中から選んで下さい。それ以外のメッセージは、ver.2.20と同じです。

HELP

内部コマント

機能　MSX DOS の機能についてオンラインヘルプを提供します。

書式　HELP [項目]

解説

パラメータを指定しないとヘルプで利用できる標準項目のリストが表示されます。これには標準のコマンドと主なシステムの機能が含まれます。

項目が指定されると、この項目についての機能解説が「ヘルプファイル」から画面に出力されます。

ヘルプファイルは、「.HLP」という拡張子がついたファイル名です。デフォルトではMSX DOSの標準のブートディスクのHELPというディレクトリ中にあります。

HELPという環境変数は最初、HELPディレクトリを参照するように設定されています（環境変数については8章を参照）。これは必要に応じて任意の他のディレクトリやディスクを参照するように、SETコマンドを使用して変更することができます。

また、ユーザーはHELPディレクトリ中に適当な.HLPファイルを追加するだけで、任意のHELP項目を付け加えることができます。ヘルプファイルはTYPEコマンドで/Pスイッチを指定して表示したのと、ほぼ同様に表示されます。

v2.30以降では、環境変数KHELPを追加しました。HELPコマンド実行時に、画面モードに応じてANKモードならばHELPで指定されるディレクトリを、漢字モードならばKHELPで指定されるディレクトリを自動的に選択します。それぞれデフォルトはMSX-DOSが起動されたドライブのルートディレクトリ中のHELP、KHELPというディレクトリになっています。

文例

```
A>HELP
```

標準ヘルプ画面を表示します。これは標準のコマンドとMSX DOSの主な機能を含んだヘルプで利用できる項目を表示します。ユーザーが追加したものはここでは表示されません。

```
A>HELP XCOPY
```

XCOPYコマンドについてのヘルプ情報を表示します。これにはコマンドの使用法と利用できるオプションについての記述が含まれます。

```
A>HELP ME
*** File for HELP not found
```

```
A>HELP ME
*** HELPファイルが見つかりません
```

このコマンドによってHELPはヘルプテキストのある場所でME.HLPというファイルを検索したが、それが見つからなかったためエラーメッセージを出力しました。ヘルプテキストを含むファイルは通常MSX-DOSが起動されたドライブの¥HELPというディレクトリ中にあり、必要ならば任意の他のヘルプファイルを加えることができます。ME.HLPが追加されていればHELP MEによってME.HLPの内容が画面に表示されます。

IF

2.31

内部コマンド

機　能	条件判断をしてコマンドを実行します。

書　式	IF [NOT] 条件 コマンド

解　説

条件が真のときにコマンドが実行されます。

IF コマンドでは以下の条件が使えます。

- EXIST ファイル名
  ファイル名が存在するときに真になります。
- 文字列 1==文字列 2
  文字列 1 と文字列 2 が等しいときに真となります。大文字と小文字は同じとみなされます。「%パラメータ」と「%環境変数%」は変換された後に比較されます。

NOT をつけると条件が成立しないときにコマンドが実行されます。

このコマンドは、version 2.31 から追加されました。

文　例

```
A>IF EXIST AUTOEXEC.BAT ECHO I have AUTOEXEC.BAT
I have AUTOEXEC.BAT
```

AUTOEXEC.BAT というファイルが存在した場合、上記のように「I have AUTOEXEC.BAT」と表示します。

```
A>IF %PROMPT%==ON ECHO Prompt is ON
Prompt is ON
```

環境変数PROMPT が「ON」の場合、上記のように「Prompt is ON」と表示します。

KMODE

外部コマンド

機 能

漢字モードを設定、解除します。

書 式

KMODE 数値 | OFF

または

KMODE [数値 | OFF] /S [d:]

解 説

漢字モードの設定を行います。数値は0～3で、その意味はDisk BASICのCALL KANJIと同様です。詳しくは11章「日本語処理」のCALL KANJIを参照して下さい。OFFはANKモードの指定です。

KMODEコマンドでは以下のスイッチが使えます。

/S　ブートセクタにモードの情報が書き込まれ、次回からは、そのディスクでは指定されたモードでMSX DOS2が立ち上がるようになります。モードを省略したときは現在の漢字モードが使われ、この際、他のシステムのブートディスクを間違って更新してしまうことのないように、ディスクの更新の前にプロンプトが出力されます。ドライブ名を省略したときは、デフォルトドライブのディスクが対象となります。

漢字ドライバがインストールされていない場合は、以下のようなメッセージが表示されますので、BASIC環境で「CALL KANJI」を実行して下さい。

```
*** Kanji driver is not installed
*** use "CALL KANJI" under BASIC
```

文 例

```
A>KMODE 3
```

漢字モード3にします。

```
A>KMODE OFF
```

スクリーンモードをANKに戻します。

```
A>KMODE /S B:
Disk in drive B: will only be able to boot MSX-DOS 2
Press any key to continue...
```

```
A>KMODE /S B:
ドライブ B: のディスクは日本語 MSX-DOS2 しか立ち上げられな
くなります.
何かキーを押して下さい
```

メッセージを出して確認した後、ドライブＢのディスクが立ち上がるときに現在の漢字モード（この場合ANK）になるようにブートセクタを書き換えます。

MD

内部コマンド

機　能

MKDIR コマンドと同じです。MKDIR コマンド（P.115）を参照して下さい。

MKDIR

内部コマンド

機能

新しいサブディレクトリを作成します。

書式

MKDIR [d:] パス

または

MD [d:] パス

解説

パス中の最後の要素が、カレントあるいは指定のドライブで作成すべき新しいサブディレクトリの名前です。これがパス中の唯一の要素なら、新しいディレクトリとしてカレントディレクトリ中に作成されます。新しいディレクトリを不可視にしたい場合は、ATDIR コマンドを使用して別に行わなければなりません。

サブディレクトリのディレクトリが表示されるときの最初の2つの項目は、常に「.」、「..」と呼ばれる特殊なディレクトリです。これらは新しいディレクトリが作成されるときに自動的に作成され、これにより、パス名中で「.」、「..」によって、それぞれカレントディレクトリ、親ディレクトリを指定できます。パスの記述については、5.1「この章の表記法」を参照して下さい。

MD コマンドは MKDIR コマンドの短縮形であり、簡便さと MS-DOS との互換性のために提供されています。

文例

```
A>MKDIR UTIL
```

UTIL というディレクトリをカレントドライブのカレントディレクトリに作成します。

```
A>MKDIR A:¥UTIL¥RAM
```

RAM というディレクトリをドライブ A のルートディレクトリにある UTIL ディレクトリ中に作成します。

MODE
内部コマンド

機　能　画面上の1行の文字数を変更します。

書　式　MODE 数値

解　説　数値は1～80までの範囲でなければならず、画面の1行ごとの文字数はその数値に設定されます。このコマンドを実行すると画面はクリアされ、カーソルはホームポジション（左上隅）に移動します。

文　例

```
A>MODE 80
```

画面を80桁モードに設定し、クリアします。

```
A>MODE 25
```

画面を25桁モードに設定し、クリアします。

MOVE

内部コマンド

機 能

ファイルを同一ディスク上で別のディレクトリに移します。

書 式

MOVE [/H][/P] 複合ファイルスペック [パス]

解 説

「複合ファイルスペック」は移動すべきファイルを指定します。

「パス」はファイルの移動先のディレクトリを指定し、これが指定されないとカレントディレクトリが使用されます。「パス」は「複合ファイルスペック」中の各ファイルスペックで参照されるドライブのそれぞれに存在していなければなりません。

特定のファイルが指定のあるいはカレントディレクトリ中に移動できない場合（例えば同一の名前のファイルがすでに存在するような場合）にはそのファイル名がエラーメッセージとともに表示され、移動処理は次のファイルに進みます。

MOVE コマンドでは以下のスイッチが使えます。

/H　不可視属性のファイルも移動させます。

/P　出力を1画面ごとに停止させます。
　　多くのエラーが起こる場合にこのスイッチを使います。

文 例

```
A>MOVE FILE1 ¥
```

ファイル「FILE1」をカレントドライブのカレントディレクトリからカレントドライブのルートディレクトリへ移動します。

```
A>MOVE /H /P E:*.COM ¥
COMMAND2.COM -- Duplicate filename
```

```
A>MOVE /H /P E:*.COM ¥
COMMAND2.COM -- ファイル名が重複しています
```

ドライブEのカレントディレクトリ中の「*.COM」に適合するすべてのファイル（不可視ファイルもそうでないファイルも）をそのドライブのルートディレクトリに移動します。

ファイル COMMAND2.COM はすでにルートディレクトリに存在するためエラーが出力され、どちらの COMMAND2.COM も移動も変更もされませんでした。

/P スイッチが指定されているので、このようなエラーが多く起こっていた場合には、画面が一杯になった段階でプロンプトが出力されて、キー入力を待ちます。

```
A>MOVE ¥UTIL¥*.COM+¥UTIL¥*.BAT
```

カレントドライブ上の「UTIL」というディレクトリ中の「*.COM」あるいは「*.BAT」に適合するすべてのファイルをそのドライブのカレントディレクトリに移動します。

MVDIR

内部コマンド

機　能	ディレクトリを同一ディスク上で別のディレクトリに移動します。
書　式	MVDIR [/H][/P] 複合ファイルスペック [パス]

解　説

「複合ファイルスペック」は移動すべきディレクトリを指定します。

2番目のパラメータ（「パス」）では移動先のディレクトリを指定し、これが指定されないとカレントディレクトリが使用されます。「パス」は「複合ファイルスペック」中の各ファイルスペックで参照されるドライブのそれぞれに存在していなければなりません。

特定のディレクトリが指定のあるいはカレントディレクトリへ移動できない場合（例えば同一の名前のディレクトリがすでに存在するような場合）には、そのディレクトリ名がエラーメッセージとともに表示され、移動処理は次のディレクトリに移ります。

ディレクトリを下位ディレクトリへ移動する（そうするとサブディレクトリのツリー構造が矛盾します）ことはできないことに注意して下さい。これを実行しようとするとエラーとなります。

MVDIR コマンドでは以下のスイッチが使えます。

/H　不可視ディレクトリも移動させます。

/P　1画面ごとに出力を停止させます。
　　エラーが数多く起こる場合にこのスイッチを使います。

文　例

```
A>MVDIR COM UTIL
```

「COM」というディレクトリおよびその下にあるすべてのディレクトリとファイルを「UTIL」というディレクトリ中に移動します。この場合、どちらのディレクトリもカレントドライブのカレントディレクトリにあります。

```
A>MVDIR ¥COM+¥BAT ¥UTIL
```

「COM」というディレクトリと「BAT」というディレクトリ、およびそれらの内容を「UTIL」というディレクトリ中に移動します。

```
A>MVDIR E:DIR?/H/P ALL
DIR2 -- Duplicate filename
```

```
A>MVDIR E:DIR?/H/P ALL
DIR2 -- ファイル名が重複しています
```

ドライブEの「DIR?」に適合する（例えばDIR1、DIR2、DIR3など）すべてのディレクトリ（これらはこの場合不可視のものであってもかまいません）、およびそれらの内容を「ALL」というディレクトリ中に移動します。「DIR2」というディレクトリは「ALL」中にすでに存在するためエラーが出力されました。どちらの「DIR2」ディレクトリもまったく影響を受けません。

PATH

内部コマンド

機　能

COM および BAT ファイル検索パスを表示・設定します。

書　式

PATH [[+ | -][d:] パス [[d:] パス [[d:] パス...]]]

解　説

パラメータが指定されないと、現在の検索パス設定がセミコロン (;) で区切られて表示されます。

「+」あるいは「-」が指定されないと、検索パスは指定のパス名のリストに設定され、既存の検索パスは削除されます。

パスのリストの前に「-」を与えると、リスト中のそれぞれのパスが現在設定されている検索パスから削除され、指定のパスが存在していない場合にはエラーとなります。

パスのリストの前に「+」を与えると、指定のそれぞれのパスが、まず(存在すれば)現在設定されている検索パスから削除され、それからその最後に追加されます。これによって検索パス中のパスの順序を変更することができ、また現在の検索パスの終りに新しいパスを追加することができます。+の構文を使用してひとつのコマンドで与えられるものよりも長い検索パスを設定することができます。検索パスの最大長は 255 文字であるのに対して、コマンドの最大長が 127 文字であるため、コマンドラインからは 127 文字しか入力できません。

COM または BAT ファイルを検索する場合には、現在の検索パス中のパスが左から右へ順に使用されます。

検索パス中のパスはドライブを含んだルートディレクトリから始まる完全なパスで指定することが望まれます。そうしないとカレントドライブやディレクトリが変わったときに、検索パスの意味が変わる可能性があります。

検索パスは環境変数として保存されるので (8章「環境変数の設定」を参照)、SET コマンドによっても表示・設定することができます。

文　例

```
A>PATH E:¥COM E:¥BAT
```

COM あるいは BAT ファイルが次に検索される場合、検索されるディレクトリはカレントドライブのカレントディレクトリ、ドライブ E のルートディレクトリ中の「COM」ディレクトリ、ドライブ E のルートディレクトリ中の「BAT」ディレクトリという順になります。

```
A>PATH
;E:¥COM; E:¥BAT
```

パラメータが指定されなかったので、現在の検索パスが表示されました。

```
A>PATH +A:¥COM;A:¥BAT
```

「A:¥COM」および「A:¥BAT」というディレクトリが検索パスの最後に追加されます。

```
A>PATH
;E:¥COM; E:¥BAT; A:¥COM; A:¥BAT
```

新しい検索パスが表示されます。

```
A>PATH -E:¥COM,E:¥BAT
```

「E:¥COM」および「E:¥BAT」ディレクトリを現在の検索パスから削除します。

```
A>PATH
;A:¥COM; A:¥BAT
```

新しい検索パスをまた表示します。

PAUSE

内部コマンド

機 能 バッチファイル中でプロンプトを表示しキーの入力待ちにする。

書 式 PAUSE [コメント]

解 説 コメントは任意の文字のシーケンスで構成されます。

もしコメントが与えられるとそれが表示され、続いて「Press any key to continue」「何かキーを押して下さい.」というプロンプトが出力されます。システムはキーが押されるのを待ち、それが印字可能な文字の場合には押されたキーを表示します。パラメータとしてコメントが与えられないと、プロンプトだけが表示されます。

このコマンドは主にバッチファイル中からプロンプトを出力するために使用されます。

ver.2.30から、PAUSEコマンドのプロンプトが短くなりました。SCREEN 1のデフォルト幅（WIDTH29）で1行におさめるためです。

旧

```
Press any key to continue...
```

新

```
Press any key to continue
```

文 例

```
A>PAUSE
Press any key to continue
```

```
A>PAUSE
何かキーを押して下さい.
```

コメントを指定していないのでプロンプトだけが表示されました。

```
A>PAUSE Insert document disk in drive B:
Insert document disk in drive B:
Press any key to continue
```

```
A>PAUSE Insert document disk in drive B:
Insert document disk in drive B:
何かキーを押して下さい.
```

「Insert document diskin drive B:」というコメントを指定したので、これがプロンプトの前に表示されています。

RAMDISK

内部コマンド

機　能	RAM ディスクの大きさを表示、あるいは設定します。
書　式	RAMDISK [数値 [K]][/D]

解　説

パラメータを指定しないと、現在のRAMディスクに割り当てられているメモリの大きさがキロバイト単位で表示されます。

数値が与えられると、新しいRAMディスクの「最大サイズ」の指定となります。単位はキロバイトで指定します。範囲は0~4064です。RAMディスクは常に16Kバイトの倍数で作成されるため、この数値はもっとも近い16Kバイトの倍数に切り上げられます。

RAMDISK コマンドでは以下のスイッチが使えます。

/D　RAMDISK が削除されます。数値0が与えられる場合も、RAMディスクが削除されます。

RAMディスクに利用できるメモリがまったくない場合には「notenough memory」「メモリー不足です」エラーとなりますが、サイズに見合うだけの未使用メモリがない場合には、指定の最大サイズ内で最大のRAMディスクが作成されます。

指定の数値はRAMディスクに使用するためのRAMの最大量です。実際には、システムが割り当てられたRAMの一部をFATやディレクトリなどの用途で使用するため、新たに作成されるRAMディスクで利用できる未使用領域の最大値とは異なります。

使用可能な RAMDISK の容量は、RAM128K バイトの MSX マシンでは最大32Kバイト、RAM256Kバイトの turboR マシンでは通常最大96Kバイトです。

新しいRAMディスクが作成される前にRAMディスクがすでに存在する場合には、

```
Destroy all data on RAM disk (Y/N)?
```

```
RAM ディスク上の全てのデータを消去しますか (Y/N)?
```

というプロンプトが表示され、データを誤って消去してしまわないように注意を促します。

/D を指定すると自動的に既存のRAM ディスクは削除され、プロンプトは表示されません。

RAM ディスクを作成すると、ドライブ H として参照することができます。

RAMDISK コマンドは通常 AUTOEXEC.BAT バッチファイル中だけで使用し、できる限り大きなRAM ディスクを作成するように大きな数値を指定するとよいでしょう。

またRAM ディスクは停電などのコンピュータへの電源の障害やリセットで失われるため、フロッピーディスクに保存されていないデータを RAM ディスクに記録する場合には注意が必要です。

文 例

```
A>RAMDISK
RAMDISK=16K
```

パラメータを指定しなかったため現在のサイズが表示されます（この場合は 16K)。

```
A>RAMDISK
*** RAM disk does not exist
```

```
A>RAMDISK
*** RAM DISK がありません
```

パラメータを指定しなかったが、RAM ディスクは作成されていなかったためエラーとなりました。

```
A>RAMDISK = 32
Destroy all data on RAM disk (Y/N)?
```

```
A>RAMDISK = 32
RAM ディスク上の全てのデータを消去しますか (Y/N)?
```

RAM ディスクがすでに存在するためプロンプトが表示されました。ここで [Y] を押すと、現在の RAM ディスクは削除され、新しい RAMDISK

が最大サイズ32Kバイトの大きさで作成されます。[N] を押すと、新しいRAMDISKは作成されません。

RD

内部コマンド

機　能

RMDIR コマンドと同じです。RMDIR コマンド（P.133）を参照して下さい。

REM

内部コマンド

機能 バッチファイル中にコメントを入れます。

書式 REM [コメント]

解説 コメントは無視され、次のコマンドが実行されます。コメントはコマンド行の最大長（127文字）までの任意の文字のシーケンスで構成されます。

文例

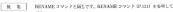
```
A>REM This is my AUTOEXEC batch file
```

このコマンドはバッチファイル中にあっても、キーボードからコマンドとして入力しても何も行いません。

REN

内部コマンド

機　能

RENAME コマンドと同じです。RENAME コマンド（P.131）を参照して下さい。

RENAME

内部コマンド

機　能　　ひとつあるいは複数のファイル名を変更します。

書　式　　RENAME[/H][/P] 複合ファイルスペック ファイル名

または

REN [/H][/P] 複合ファイルスペック ファイル名

「複合ファイルスペック」で名前を変更するファイルを指定します。

2番目の「ファイル名」はファイルに対しての新しい名前を指定します。新しい名前の中で「?」を指定すると、変更を受けるファイル名と対応する文字が当てはめられるので、ワイルドカードを含むファイル名（P.60のファイル名についての記述を参照）で変更が可能となります。2番目の「ファイル名」の主ファイル名もしくは拡張子に「*」を使用すると、これはすべての文字を「?」と指定したものと同等なため、ファイル名あるいは拡張子全体がそのまま変更されずに残ります。

何らかの理由で特定のファイルの名前が変更できない場合、例えば、読み出し専用の場合や、新たに指定した名前と同じ名前のファイルまたはディレクトリがすでに存在する場合、そのファイル名がエラーメッセージとともに表示され、名前の変更の処理は次のファイルへ進みます。

RENAME コマンドでは以下のスイッチが使えます。

/H　不可視属性のファイルも名前を変更することができます。

/P　1画面ごとに出力を停止させることができます。
エラーが数多く起こる場合にこのスイッチを使います。

```
A>RENAME FILE1 FILE2
```

カレントドライブのカレントディレクトリのFILE1というファイルをFILE2に変更します。

```
A>RENAME B:¥DIR1¥*.DOC/H/P *.OLD
FILE2.DOC -- Duplicate filename
```

```
A>RENAME B:¥DIR1¥*.DOC/H/P *.OLD
FILE2.DOC -- ファイル名が重複しています
```

ドライブBのルートディレクトリ中のDIR1というディレクトリ中の「*.DOC」に適合するすべてのファイル（不可視ファイルを含む）を同じ主ファイル名で拡張子が.OLDのファイル名に変更します。ディレクトリ中にすでにFILE2.OLDというファイルが存在したため、ファイルFILE2.DOCは名前を変更できずエラーが表示されました。FILE2.DOCもFILE2.OLDもまったく変更を受けていません。/Pが指定されているので、エラーが多く出力された場合、1画面ごとに停止してキー入力を待ちます。

```
A>RENAME DOC + FILE1 *.OLD
```

DOCというディレクトリ中のすべてのファイルおよびファイルFILE1（どちらもカレントドライブのカレントディレクトリにある）のファイル名を拡張子.OLDを持つものに変更します。

RMDIR

内部コマンド

機能	ひとつあるいは複数のサブディレクトリを削除します。

書式

RMDIR [/H][/P] 複合ファイルスペック

または

RD [/H][/P] 複合ファイルスペック

解説

「複合ファイルスペック」で削除すべきディレクトリを指定します。

ディレクトリを削除するためには、常にディレクトリ中に含まれる特殊な「.」および「..」という特殊なディレクトリ以外にファイルやディレクトリを含んでいてはなりません。これらの特殊なディレクトリは新しいディレクトリが作成されるとき自動的に置かれ、削除することはできません。

これらは新しいディレクトリが作成されるときに自動的に作成され、これにより、パス名中で「.」、「..」によって、それぞれカレントディレクトリ、親ディレクトリを指定できます（パスの記述については、5.1「この章の表記法」を参照）。

ディレクトリが何らかの理由で削除できない（例えばそれが空でない）場合、そのディレクトリの名前がエラーメッセージとともに表示され、削除処理は次のディレクトリに進みます。

RMDIR コマンドでは以下のスイッチが使えます。

/H　不可視属性のディレクトリも削除できるようになります。

/P　1画面ごとに出力を停止させることができます。
　　多くのエラーが起こる場合はこのスイッチを使います。

文例

```
A>RMDIR DIR1
```

カレントドライブのカレントディレクトリ中の DIR1 というディレクトリを削除します。

```
A>RMDIR B:¥COM + B:¥BAT
```

ディレクトリ COM および BAT をドライブ B のルートディレクトリから削除します。

```
A>RMDIR ¥*.*
UTIL -- Directory not empty
```

```
A>RMDIR ¥*.*
UTIL -- ディレクトリが空ではありません
```

カレントドライブのルートディレクトリからすべてのディレクトリを除去しようとしたが、UTILというディレクトリは空でなく、そのためエラーが表示されました。UTILとその内容にはまったく影響がありません。

RNDIR

内部コマンド

機能 | ひとつあるいは複数のサブディレクトリの名前を変更します。

書式 | RNDIR [/H][/P] 複合ファイルスペック ファイル名

解説 | 「複合ファイルスペック」で名前を変更するディレクトリを指定します。ディレクトリの内容は変更されません。

2番目の「ファイル名」はディレクトリの新しい名前を指定します。新しい名前の中で「?」を指定すると、変更を受けるディレクトリ名の対応する文字が当てはめられ、ワイルドカードを含むディレクトリ名（P.60のファイル名についての記述を参照）で変更が可能となります。2番目の「ファイル名」で「*」を使用すると、これはすべての文字を「?」と指定したものと同等なため、ディレクトリ名のファイル名あるいは拡張子全体がそのまま変更されずに残ります。

何らかの理由で特定のディレクトリの名前が変更できない場合、例えば新たに指定した名前と同じ名前のファイルまたはディレクトリがすでに存在する場合、そのディレクトリ名がエラーメッセージとともに表示され名前の変更の処理は次のディレクトリへ進みます。

RNDIR コマンドでは以下のスイッチが使えます。

/H 不可視属性のディレクトリの名前も変更します。

/P 出力を1画面ごとに停止させます。
多くのエラーが起こる場合にこのスイッチを使います。

文例

```
A>RNDIR UTIL COM
```

カレントドライブのカレントディレクトリのUTILというディレクトリの名前をCOMに変更します。

```
A>RNDIR A:¥DIR?.*/H/P *.DLD
DIR1.DLD -- Duplicate filename
```

```
A>RNDIR A:¥DIR?.*/H/P *.OLD
DIR1.OLD -- ファイル名が重複しています
```

ドライブAのルートディレクトリ中のDIR?.*にマッチするすべてのディレクトリ（不可視ディレクトリもそうでないディレクトリも）を.OLDという拡張子を持つものに名前を変更します。DIR1.OLDというディレクトリがすでに存在したため、ディレクトリDIR1の名前は変更できず、エラーが表示されました。エラーが数多く起こる場合には、/Pによって1画面ごとに表示を停止します。

```
A>RNDIR COM + BAT *.OLD
```

COMおよびBATというディレクトリをそれぞれCOM.OLD、BAT.OLDに名前を変更します。

SET

内部コマンド

機　能

環境変数を表示・設定します。

書　式

SET [名前][セパレータ][値]

解　説

パラメータが指定されないと現在定義されているすべての環境変数とその値が表示されます。初期状態ではデフォルトの値に設定されたいくつかの項目があります（8章「環境変数の設定」を参照）。

「名前」だけが指定されると、その環境変数の現在の値が表示されます。

「名前」の後に「セパレータ」が指定された場合、「セパレータ」に続く値が名前に設定されます。値が与えられないとその環境変数は環境領域から削除されます。

環境変数に使用されるメモリ領域はディスクバッファとしても使用されます。したがって SET コマンドを使用していて「not enough memory」「メモリー不足です」エラーが起こった場合には、ディスクバッファの数を減らすことで対処できることがあります（P.76「BUFFERS」を参照）。

8章では環境変数についてのより詳細な情報やデフォルトで設定されている値などが記述されています。

文　例

```
A>SET
REDIR=ON
UPPER=OFF
ECHO=OFF
PROMPT=OFF
PATH=;
TIME=12
DATE=yy-mm-dd
TEMP=A:¥
KHELP=A:¥KHELP …… ・ver.2 30から追加された環境変数
HELP=A:¥HELP
```

パラメータを指定しなかったため、すべての現在設定されている環境変数が表示されます（この場合は、デフォルトの値）。

```
A>SET HELP=A:¥HELP
```

環境変数 HELP の値を A:¥HELP に設定します。

```
A>SET HELP
A:¥HELP
```

HELPの現在の値を表示します。

```
A>SET HELP=
```

環境変数HELPの値を設定しないと、環境変数のリストから環境変数HELPを除去します。

TIME

内部コマンド

機　能

現在の時刻を表示、あるいは設定します。

書　式

TIME [時刻]

解　説

コマンドの後に「時刻」が与えられると時刻はその値に設定されます(形式は後述)。コマンドの後に「時刻」が与えられないと現在の時刻が表示され、新しい時刻の入力待ちとなります。入力が与えられないと(⏎キーだけが押されると)、現在の時刻は変更されません。入力が与えられるとその入力は新しい時刻であると見なされ、後述のように解釈されます。時刻が不正である場合にはエラーメッセージが表示され、再び新しい時刻の入力待ちになります。

「時刻」は4組までの数値から成り、それぞれは下記の区切り文字で区切ります。

空白　タブ　-　/　:

これら文字のどちらの側にも空白が来てかまいません。入力を省略された数値は現在の設定が使用されます。最初の数値は時刻、2番目が分、3番目が秒、4番目が1/100秒です。しかし、1/100秒については現在の値を知ったり、新たな値を入力することはそれほど役立つことではないため表示はされません。

時刻が表示される形式は固定されておらず、変更することができます。TIMEという環境変数(環境変数については8章を参照)はデフォルトでは値「12」に設定されており、これは時刻が12時間形式で表示され、午前と午後を表すのに後に「a」や「p」をつけることを示しています。

コマンドSET TIME 24によって、時刻は24時間モードで表示されます。時刻はどちらの形式でも入力することができます。時刻の形式はDIRコマンドによって表示される時刻にも影響を与えます。

文　例

```
A>TIME 16:45
```

現在の時刻を午後4:45に設定します。

```
A>TIME
Current time is 10:45:00a
Enter new time:
```

```
A>TIME
現在の時刻は 10:45:00aです
新しい時刻を入力して下さい:
```

パラメータを与えなかったため、現在の時刻が表示され（この場合は12時間モードで）、新しい時刻のプロンプトが出力されました。

```
A>TIME 10-50-30
```

時刻を午前10時50分30秒に設定します。

TYPE

内部コマンド

機能

ファイルまたはデバイスからデータを表示します。

書式

TYPE [/H][/P][/A][/B] 複合ファイルスペック | デバイス

解説

「複合ファイルスペック」は表示するファイルを指定します。「複合ファイルスペック」にワイルドカードを使用すると、それぞれのファイルが表示される前にそのファイル名が出力されます。

TYPE コマンドでは以下のスイッチが使えます。

/H 不可視属性のファイルも表示します。

/P 出力が1画面ごとに停止し、キーが押されるのを待ちます。

/A アスキーファイルとして扱います（デフォルトです)。

/B バイナリファイルとして扱います。

ファイルの終りに達するまでデータがそれぞれのファイルから読まれて画面上に変更なしに表示されます。ファイルがコントロール文字を含む場合は、画面上でおかしな動作を見せることがあります。

/B が指定されないと、TYPE はエンドオブファイル文字（^Z）を探しそれを見つけると停止します。また、復帰、改行、およびタブを除くコントロール文字は表示可能な文字（例えば、^A なら^と A という2文字、^W なら^と W という2文字）に変換されます。

文例

```
A>TYPE FILE1
```

FILE1 というファイルを最初のエンドオブファイル文字まで画面に表示します。

```
A>TYPE *.BAT/H/P
```

不可視であるものも含めすべてのバッチファイルを読み込んで表示します。1画面ごとにプロンプトが表示されて停止します。

```
A>TYPE AUTOEXEC.BAT + REBOOT.BAT
```

ファイル AUTOEXEC.BAT および REBOOTBAT を表示します。

```
A>TYPE /B DIR1
```

ディレクトリ DIR1 中のすべてのファイルを、ファイル中のデータを変換を行わずに画面に表示します。

UNDEL

外部コマント

機能	以前に削除されたファイルを復活します。

書式	UNDEL [ファイルスペック]

解説

「ファイルスペック」には復活したいファイルもしくはディレクトリを指定します。指定されなかった場合は*.*となります。

ファイルはMSX DOS2フォーマットのディスクでMSX DOS2を使用して削除されたもので、ファイルもしくはディレクトリが削除されてからそのディスクに何か書き込みを行う前であれば復活できます。

削除されたディレクトリおよびその下の削除されたファイルなどを復活するには、まずディレクトリをUNDELし、次にその下のファイルなどをUNDELします。このとき、すべてのファイルなどをUNDELする場合はディレクトリ名のみを指定すればよく、「¥*.*」は省略できます。

文例

```
A>UNDEL B:HELP.MAC
```

ドライブBのカレントディレクトリからファイルHELP.MACを復活します。

```
A>UNDEL A:¥DIR1
DIR1          (sub-directory)
```

```
A>UNDEL A:¥DIR1
DIR1          (サブディレクトリ)
```

DIR1を復活します。

```
A>UNDEL A:¥DIR1¥*.*
FILE1
FILE2
```

```
A>UNDEL A:¥DIR1
FILE1
FILE2
```

DIR1中のすべての復活可能なファイルおよびディレクトリを復活します。この場合、「FILE1」と「FILE2」の2つのファイルが復活できたことになります。

VER

内部コマンド

機能

システムのバージョン番号を表示します。

書式

VER

解説

MSX-DOS ディスクシステムの3つの主なシステムプログラムのバージョン番号が表示されます。どのバージョン番号も3つの数字で構成されます。

最初の数字はMSX DOS の主バージョン番号で、MSX DOS2 の場合は常に2です。

2番目の数字はバージョン番号で、例えば重要な機能が追加された場合は変更されます。

最後の数字はリリース番号で、マイナーチェンジ、改良や修正が行われた場合に変ります。

文例

```
A>VER
MSX-DOS Kernel version 2.30
MSXDOS2.SYS version 2.30
COMMAND2.COM version 2.30
Copyright 1990 ASCII Corporation
```

ディスクシステムのバージョン番号を出力します。

VERIFY

内部コマンド

機　能	現在のディスクの書き込みベリファイの状態を表示・設定します。

書　式	VERIFY [ON	OFF]

解　説

パラメータを与えない場合、現在のベリファイの状態が画面上に表示されます。

ON あるいは OFF を指定すると、ベリファイの状態が変更されます。

ベリファイの状態はディスクへのすべての書き込みに影響を与えます。OFF の状態ではデータは単純に書き込まれます。

ON の場合、データが書き込まれた後でそれが読み込まれ、元のデータと比較され、正しく書き込まれたかどうかを確認します。このオーバーヘッドによって、ベリファイが ON の場合には書き込みは遅くなります。

この機能はディスクドライバに依存するため、ドライバがベリファイ機能をサポートしていない場合は意味を持ちません。

文　例

```
A>VERIFY
VERIFY=OFF
```

パラメータを与えなかったため、現在のベリファイの状態（この場合は OFF）が表示されます。

```
A>VERIFY ON
```

ディスク書き込みベリファイをオンにします。

VOL

内部コマンド

機能　[illegible]ューム名を表示・変更します。

書式　VOL [d:][ボリューム名]

解説　パラメータを指定しない場合、もしくはドライブ名だけを指定した場合、カレントドライブあるいは指定のドライブのボリューム名が表示されます。

「ボリューム名」を指定すると、カレントドライブあるいは指定のドライブのボリューム名が、指定のボリューム名に変更されます。

「ボリューム名」は最大、半角 11 文字もしくは全角 5 文字です。

文例

```
A>VOL B:
Volume in drive B: has no name
```

```
A>VOL B:
ドライブ B:のディスクにはボリューム名がありません
```

ドライブだけ指定したため、そのドライブ中のディスクのボリューム名が表示されます。この場合、ボリューム名は定義されていません。

```
A>VOL B:BACKUP
```

ドライブ B のボリューム名を「BACKUP」に変更します。

XCOPY

外部コマンド

機　能

ディスクのファイルおよびディレクトリを別のディスクにコピーします。

書　式

XCOPY [ファイルスペック [ファイルスペック]][/H][/T][/A][/M][/S][/E][/P][/W][/V]

解　説

XCOPYは拡張されたファイルコピーコマンドで、ファイルとディレクトリの両方を選択的にコピーすることができます（P.88「COPY」を参照）。

最初の「ファイルスペック」ではソースファイル名を指定します。

2番目の「ファイルスペック」はコピー先のファイル名です。したがってファイルはコピーを通して名前を変更することができます（標準のCOPYコマンドと同様）。

XCOPYコマンドでは以下のスイッチが使えます。

/H　不可視属性のファイルもコピーします。

/T　コピーされたファイルの日付と時間はソースファイルのものの代わりに現在のものとなります。

/A　「アーカイブ」属性がセットされているファイルだけがコピーされます。ファイルは、「不可視」属性および「読み出し専用」属性と同様にアーカイブ属性を持ちます。これはファイルが更新された（書き込まれた）時はいつでも設定されます。

/M　/A と同様ですが、ファイルをコピーした後でアーカイブ属性をリセットします。したがってこのスイッチを使用すると、ファイルが更新された場合にだけファイルを別のディスクにコピーすることができ、ファイルの効率的なバックアップが可能になります。

/S　ファイルをディレクトリごとコピーします。それぞれのディレクトリ内で適合するすべてのファイルがコピーされ、さらにその下にあるサブディレクトリとその中にあるファイルもコピーされます。コピー先のディスク上にディレクトリが存在しない場合には、そのディレクトリが作成されます。コピーの条件に適合するファイルを持たないディレクトリは作成されません。

/E　/S スイッチが指定されているとき、/E スイッチが指定されると、コピーの条件に適合するファイルを持たないディレクトリであっても作成されます。

/P　それぞれのファイルをコピーする前に停止し、そのファイルをコピーして良いかどうかを聞いてきます。これによって、ファイルを選択してコピーすることができます。

/W　コピーを始める前に停止しプロンプトを出力するため、ディスクを入れ換えることができます。

/V　XCOPY コマンドの処理の間、ディスクドライバがベリファイ機能を持っていれば書き込みチェックを行うことができます（P.146「VERIFY」を参照)。これによってデータがディスクに正しく書き込まれることが保証されますが、処理にかかる時間は増加します。

文　例

```
A>XCOPY B:¥
```

ドライブBのルートディレクトリ中のすべてのファイルをカレントドライブのカレントディレクトリへコピーします。この場合は、標準で組み込まれている COPY コマンドを使用する以上のメリットはありません。

```
A>XCOPY *.* B: /H/S/M
```

不可視ファイルを含むすべてのファイルを、以前に/Mスイッチを指定したコマンドが実行されてからファイルが変更されたもの（アーカイブ属性がセットされているもの）に限り、ドライブBにコピーします。ファイルはその後、変更されていないものとしてアーカイブ属性がリセットされます。

カレントディレクトリのファイルだけでなく、ディレクトリとすべてのその下にあるサブディレクトリおよびファイルもコピーされます。

XDIR

外部コマンド

機能	ディレクトリ中のすべてのファイルのリストを表示します。
書式	XDIR [ファイルスペック][/H]

解説

「ファイルスペック」で表示するファイルを指定します。

XDIR は DIR コマンドと類似していますが、ファイルの日付と時間を表示しません。

指定のディレクトリ中のすべてのファイルがリストされると、下にあるサブディレクトリ中のファイルもインデントを付けられてリストされます。これによって、完全なディレクトリツリーあるいはディスクのファイル一覧を取ることができます。

DIR コマンドでは以下のスイッチが使えます。

/H　不可視ファイルを表示することができます。

文例

```
A>XDIR
Volume in drive A. is MSX-DDS 2
X-Directory of A:¥

 MSXDDS2 SYS      r      4B70
 CDMMAND2.COM     r     15472
 AUTDEXEC.BAT              57
 REBOOT.BAT                57
 ¥UTILS
            CHKDSK.CDM              76B0
            DISKCOPY.COM            716B
            FIXDISK.CDM              76B
            UNDEL.CDM               396B
            XCDPY.CDM              10112
            XDIR.COM                716B
            MKSYS.BAT                569
            AUTDEXEC.BAT              47
            REBDDT.BAT                90
 ¥HELP
            ASSIGN.HLP               B19
            ATDIR.HLP               1527
            ATTRIB.HLP              1828
                 .
                 .
                 .
292K in 117 files   530K free
```

```
A>XDIR
ドライブA:のボリューム名は MSX-DOS2
拡張ディレクトリ A:¥

 MSXDOS2.SYS     r     4870
 COMMAND2.COM    r    15472
 AUTOEXEC.BAT           154
 REBOOT.BAT              99
 ¥UTILS
            CHKDSK.COM              7680
            DISKCOPY.COM            7168
            FIXDISK.COM              768
            UNDEL.COM               3968
            XCOPY.COM              10112
            XDIR.COM                7168
            MKSYS.BAT                569
            AUTOEXEC.BAT              47
            REBOOT.BAT                90
 ¥HELP
            ASSIGN.HLP               819
            ATDIR.HLP               1527
            ATTRIB.HLP              1828
               .
               .
               .
292Kバイトが計117ファイルで使用中
356Kバイトが使用可能
```

カレントドライブのカレントディレクトリから下のディレクトリをすべて表示します。

```
A>XDIR B:¥DIR1
```

ディレクトリDIR1中のすべてのファイルとサブディレクトリの内容を表示します。

```
A>XDIR ¥*.COM/H
```

「*.COM」に適合する不可視ファイルを含むすべてのファイルの名前を表示します。

6章 リダイレクションとパイプ

COMMAND2.COMは、以下に述べるリダイレクションおよびパイプの機能を提供しています。ただし環境変数「REDIR」を「SET REDIR=OFF」コマンドなどで「OFF」にセットすると、この機能は働かなくなるので、MSX-DOS1やCP/Mとの互換性を保つことができます。

6.1 リダイレクション

大部分のコマンドはCP/MプログラムもMSX-DOSプログラムも「標準出力」へ書き出すことによって画面にテキストを出力し、「標準入力」から読むことによってキーボードから読み込みます。しかしCOMMAND2.COMはコマンドを実行している間だけ、標準入力と標準出力を他のMSX-DOSデバイスやディスク上のファイルを参照するように、変更する機能を提供しています。これは、コマンド行にリダイレクション記号「<」、「>」、および「>>」のうちの1つ以上を含めて、その後にファイル名を置くことによって可能になります。

例えば、ECHOコマンドは標準出力に文字を出力することによって、文字列を画面に表示するだけですが、その出力を次のようにリダイレクトすることによって、その代わりにプリンタに出力を行うことができます。

```
ECHO text > PRN
```

これによってECHOコマンドの実行中に標準出力がデバイスPRNを参照するように変更できます。

```
ECHO text > file1
```

同様に、このコマンドによってFILE1という名前のファイルが作成され、ECHOコマンドの出力がファイルへ書き出されます。また、「>」記号の代わりに「>>」記号を使うと、コマンドの出力を既存のファイルに追加することができます。指定されたファイルが存在しない場合にはファイルを作成します。

標準入力を変更するには、「<」記号を「>」記号と同様の方法で使用します。この場合ファイルは既存のものでなければならず、コマンドに対して適切な入力が含まれていなければなりません。コマンドがファイルの終わりを越えて入力を読もうとすると、続行できないためコマンドは打ち切られます。

コマンド行にリダイレクション情報が指定されると、それはCOMMAND2.COMがリダイレクションをセットアップするために使い、コマンド行からは除去されます。したがって上記の例では、ECHOコマンドはリダイレクション記号やファイル名をエコーしません。

バッチファイルの入力や出力がリダイレクトされると、リダイレクションはバッチファイル中のすべてのコマンドに対して適用されます。ただしバッチファイル中の個々のコマンドでリダイレクションを使用することも可能で、この場合バッチファイルに対してのリダイレクションに優先します。バッチファイル中のコマンドについて詳しくは7章「バッチファイル」を参照して下さい。

6.2　パイプ

コマンドやプログラムの入出力を、他のデバイスやディスクファイルに対してリダイレクトできるのと同様、ひとつのコマンドの標準出力を別のコマンドの標準入力へリダイレクト、すなわち「パイプ」ができます。

一般には、2番目のコマンドはそれ自身の標準入力から読み込んでデータを修正し、それをそれ自身の標準出力へ書き出すプログラムです。このようなプログラムは「フィルタ」と呼ばれます。例えば、フィルタは標準入力からデータを読んで、それをアルファベット順にソートし、それを標準出力に書き出すといったことができます。したがってDIRコマンドの出力をソートすることもできます。

パイプはコマンド行上で2つ以上のコマンドを「|」記号で区切ることによって指示します。「|」記号の左側のコマンドが最初に実行され、その出力がCOMMAND2.COMによって一時的に作成されるテンポラリファイルにリダイレクトされます。それから2番目のコマンドがその標準入力を同一のテンポラリファイルからリダイレクトされて実行されます。2番目のコマンドが終了すると、テンポラリファイルは削除されます。2番目のコマンドの標準出力を、3番目のコマンドの標準入力へパイプしたり、さらに（別のコマンドに）続けたりすることももちろんできます。

パイプを含むコマンド行で入力のリダイレクションが起こると、リダイレクションはパイプの最初のコマンドに対して適用され、その他のコマンドはその標準入力をパイプの直前のコマンドの標準出力から受け取ります。同様に、パイプを含むコマンド行で出力のリダイレクションが指定された場合には、そのリダイレクションはコマンド行の最後のコマンドに対して適用されます。

パイプをバッチファイルの入力または出力のどちらかで直接使うことはできません。しかし、COMMAND2コマンド（P.84「COMMAND2」を参照）から実行するならばバッチファイルでパイプを使用することができます。なぜなら、その場合リダイレクトされるのはバッチファイルではなくCOMMAND2コマンドだからです。

上記のように、ひとつのコマンドの出力を他の入力へパイプするためには、テンポラリファイルが COMMAND2.COM によって作成され、削除されます。これらのテンポラリファイルの場所は TEMP 環境変数(8章「環境変数の設定」を参照)によって指定され、これは他のドライブやディレクトリを参照するように変更できます。例えば TEMP が RAM ディスク上のディレクトリを参照していれば、パイプはかなり高速になります。デフォルトでは TEMP はブートディスクのルートディレクトリになっています。テンポラリファイルに使用されるファイル名は COMMAND2.COM によって作成されるため、TEMP はドライブとディレクトリだけを指定しておきます。テンポラリファイルのファイル名は次のような形式となります。

```
%PIPExxx.$$$
```

ここで xxx は 3 桁の数字で、TEMP ディレクトリ中の他のファイルと衝突しないように COMMAND2.COM によって選択されます。

7章 バッチファイル

コマンドがMSX-DOSに与えられ、それが内部コマンドでない場合、その名前で拡張子がCOMまたはBATのファイルが検索されます。カレントディレクトリ中で見つからないと、現在の検索パスが参照されます（P.121「PATH」を参照）。COMファイルが見つかると ロードして実行します。BATファイルが見つかると、MSX-DOSはバッチファイルの実行を開始します。

バッチファイルとはコマンドのリストを含むテキストファイルであり、これらのコマンドは一時に1行ずつファイルから読み込まれて、あたかもキーボードから入力されたかのように実行されます。5章「コマンド」で説明したコマンドのうち、ECHOやPAUSE、IFなどは、主としてバッチファイル中で使用するために用意されています。

それぞれのコマンドが読み込まれると、通常即座に実行されます。ただしECHOという環境変数を「SETECHO ON」コマンドを使って「ON」にセットすると、各コマンドの実行前にコマンド行自体を画面上に表示することができるようになります（8章の環境変数についての記述を参照）。コマンド行はその場合、%パラメータの代入（後述）が実行されてからエコーされます。コマンド「SET ECHOOFF」は、これを通常の状態に戻します。

バッチファイルを起動するコマンド行では、他のコマンドや外部プログラム名と同様パラメータをバッチファイルの名前の後に続けることができます。これらのパラメータは%0～%9を指定することによって、バッチファイル中のどこからでもアクセスすることができます。%1がコマンド行で指定される最初のパラメータで、%2が2番目のパラメータなどとなります。%0はバッチファイルそれ自身の名前です。%数字は元のコマンド行のパラメータで置き換えられ、バッチファイル中のどこででも使用できます。コマンド行で実際に「%」記号を使用するには、2つの%（「%%」）を与えなければならず、これは単一の%に変換されます。

また、環境変数名の前後に%を付けることにより、環境変数をバッチ中に取り込むことができます（これは、ver.2.31から追加された機能です）。%環境変数名%は環境変数の設定値に置き換えられ、バッチファイル中のどこででも（バッチファイル中に限らずコマンド行ででも）使用できます。

バッチファイル中のコマンドの実行がなんらかの理由で中断された場合（特に[CTRL]＋[STOP]や[CTRL]＋[C]が押された場合）、次のようなプロンプトが出力されます。

```
Terminate batch file (Y/N)?
```

```
バッチ処理を中止しますか (Y/N)?
```

これに対して「Y」を答えると、バッチファイル全体の実行が停止します。応答が「N」であると、バッチファイルの実行はバッチファイル中の次のコマンドから
続行されます。

MSX DOSがバッチファイル中のコマンドを実行し終ると、バッチファイル中の次のコマンドをディスクから読み出さなければならない場合があります。その場合正しいディスクがドライブ中にないとプロンプトが出力されます。例えば、バッチファイルが最初ドライブAから実行された場合、次のようなプロンプトが出力されます。

```
Insert disk for batch file in drive A:
Press any key to continue
```

```
バッチファイルの入ったディスクをドライブA:に入れて
何かキーを押して下さい.
```

正しいディスクが挿入され、キーが押されると、バッチファイルの実行は正常に続行します。

以下に示すのは非常に単純なバッチファイルであり、最初のいくつかのパラメータを表示するだけのものです。

```
ECHO Parameter 0 = %0
ECHO Parameter 1 = %1
ECHO Parameter 2 = %2
ECHO Parameter 3 = %3
```

これをMYBAT.BATとすると、コマンドMYBAT a b cは以下の出力を行います。

```
Parameter 0 = MYBAT
Parameter 1 = a
Parameter 2 = b
Parameter 3 = c
```

最初にMSX DOSが起動されると、AUTOEXEC.BATという特殊なバッチファイルが検索され、もしあれば実行されます。これにはどんなMSX-DOSコマンドを含めても良く、その中にはRAMディスクを設定するRAMDISKコマンドのように、立ち上げ時に1度だけ実行すればよい初期化コマンドなどを指定しておくと便利です。

この場合、AUTOEXEC.BATにはひとつの%パラメータが%1として渡されます。これはMSX DOSが起動されたドライブ名で、コロンが後に付いた通常のドライブ名の形を取ります。

もうひとつの特殊なバッチファイルはREBOOT.BATです。これは、DiskBASICを使用した後で、MSX DOSが再起動されるときに実行されます。AUTOEXEC.BATファイルと同様、MSX-DOSが再起動したドライブが%1パラメータとして渡されます。

最初であれ、2度目以降であれ、MSX-DOSを起動するときは、通常いくつかのコマンドを実行する必要があり、これらをREBOOTバッチファイルに入れておきます。それらは、AUTOEXECバッチファイルをコマンドREBOOT %1で終らせることによって、AUTOEXECバッチファイルから実行することができます。REBOOTバッチファイル中に入れておくコマンドの一例は、外部コマンドの検索パスを設定するPATHコマンドです。このコマンドを使用して検索パスを設定する場合、%1を使用して、どのドライブから起動しても正しいパスを設定することができます。

バッチファイル中のコマンドが別のバッチファイルの名前である場合、2番目のバッチファイルが続いて実行されます。それが終了すると、制御はコマンドインタープリタに戻り、最初のバッチファイルには戻りません。つまり、バッチコマンドは「チェイン」します。

バッチファイルを「ネスト」する、つまり上記の場合に最初のバッチファイルへ制御を戻すには、COMMAND2コマンド（P.84「COMMAND2」を参照）に、2番目のバッチファイルの名前をパラメータとして渡します。その場合、2番目のバッチファイルが終了すると最初のバッチファイルがCOMMAND2コマンドの後のコマンドから続けられます。

一般的なAUTOEXECバッチファイルの例を次に示します。

```
ECHO AUTOEXEC executing
RAMDISK 100
RAMDISK
COPY COMMAND2.COM H:¥
REBOOT %1
```

一般的なREBOOTバッチファイルの例を次に示します。

```
ECHO REBOOT executing
PATH H:¥;%1¥UTILS;%1¥BATCH
SET SHELL=H:¥COMMAND2.COM
SET TEMP=H:¥
SET PROMPT ON
H:
```

AUTOEXECバッチファイルが実行されると、「AUTOEXEC executing」のメッセージを表示し、RAMディスクを最大100Kで設定します。次に別のRAMDISKコマンドで作成されたRAMディスクの実際の大きさを表示します。そしてCOPYコマンドでCOMMAND2.COMをRAMディスク上にコピーし、再ロードを高速にできるようにします。最後にREBOOTバッチファイルを起動し、それに%1パラメータ（ブートドライブ名）を渡します。

REBOOTバッチファイルはメッセージを表示し、それからPATHを設定します。パス中の最初の項目はAUTOEXECバッチファイルで作成されたRAMディスクを参照しておりその他の項目はMSX DOSがブートされたディスク（つまり%1）中のディレクトリを参照しています。次にCOMMAND2.COMがRAMディスクから高速に再ロードできるようにSHELL環境変数をセットアップし、RAMディスク上にパイプファイルを作成するようにTEMP環境変数を設定します。最後にPROMPTをONにセットして、カレントディレクトリがプロンプトとして表示されるようにし、RAMディスクをカレントドライブにします。

8章
環境変数の設定

MSX DOSは「環境変数」のリストをワークエリア内に記憶しています。環境変数とは名前があってそれに関連した値を持ったものです。

環境変数の名前はユーザーが任意に設定することができ、ファイル名で使用できるのと同じ文字を使うことができます。最大長は255文字です。MSX-DOSはデフォルトで設定されるいくつかの環境変数を用意しています。

環境変数の値は最大長255文字までの任意の文字からなる単なる文字列です。文字についてはいかなる処理も実行されないため、大小文字の区別は保存されます。存在しないすべての環境変数はヌル値を取る（つまり文字がない）ものとされます。

環境変数はSETコマンドによって変更あるいは設定できます。また、これによって現在設定されている環境変数を表示することもできます。

デフォルトで設定されている環境変数と、それらの値の解釈を以下に示します。

なお、2.30 は、ver.2.30で追加された環境変数であることを意味します。

8.1 環境変数の説明

- ECHO

  これは、バッチファイルから読まれる行のエコーを制御します（7章のバッチファイルについての記述を参照）。「ON」（小文字も可）以外のすべての値は、「OFF」として解釈されます。

- PROMPT

  これは、コマンドレベルのプロンプトの表示を制御します。「ON」（小文字も可）以外のすべての値は「OFF」として解釈されます。

  PROMPTがOFFの場合（デフォルト）、プロンプトは「A>」のようにカレントドライブと「>」によって構成されます。

  PROMPTがONの場合、プロンプトは、「A:¥COM>」のように、カレントドライブ名とカレントディレクトリおよび「>」によって構成されます。この表示を行うために

はカレントドライブのカレントディレクトリを読むためにディスクをアクセスしなければならず、したがってプロンプトが現れるのにはその分時間がかかります。

- PATH

COMMAND2.COM が与えられたコマンドを検索するための検索パスは、環境変数 PATH として保持されており、PATH コマンドにより操作されます。

- SHELL

環境変数 SHELL はどこにコマンドインタープリタ (COMMAND2.COM) が存在するかを示し、デフォルトではそれがロードされたところにセットされています。

コマンドインタープリタが外部コマンドを実行した後、それ自身をディスクから再ロードする必要がある場合それは環境変数 SHELL を調べてそれが示すファイルから自分自身をロードしようとします。これがエラーになると、最初にロードされたドライブのルートディレクトリからロードしようとします。

コマンドインタープリタを別のドライブやディレクトリから再ロードさせるために、COMMAND2.COM をコピーして SHELL をそこから参照するように設定することができます。例えば、それをコマンド COPY COMMAND2.COM H:¥で RAMDISK にコピーしたとすると、SHELL はコマンド SET SHELL=H:¥COMMAND2.COM で設定します。

- TIME

TIME は MSX-DOS によって表示される時刻の形式を指定します。「24」(24 時間形式で表示する) でない場合には、「12」と解釈されますが、これは 12 時間形式に午前午後の表示を加えて表示するものです。

時刻の入力は、どちらの形式でも入力しても区別できますので、環境変数 TIME は無視されます。

- DATE

DATE は MSX-DOS による日付の表示と入力の形式を指定します。デフォルトではその MSX マシンが使用される国に合わせてあります。それは、日付・時刻のセパレータで区切られた 3 文字あるいは 3 組の文字の形式を取ります (P.92「DATE」を参照)。

例えばアメリカ形式にセットするには、コマンド SET DATE=MM/DD/YY を入力します。

- HELP と KHELP (2.30)

HELP コマンドにヘルプが必要なコマンド名を指定すると、HELP、KHELP 環境によって指定されるディレクトリからファイルが読み出され表示されます。

HELP コマンド実行時に、画面モードに応じて ANK モードならば HELP で指定されるディレクトリを、漢字モードならば KHELP で指定されるディレクトリを自動的に

選択します。それぞれデフォルトはMSX DOSが起動されたドライブのルートディレクトリ中のHELP、KHELPというディレクトリになっています。

- APPEND

  APPENDはデフォルトでは定義されませんが、設定されるとシステムに対して特別な意味のある環境変数となります。これは標準のCP/Mプログラムとともにだけ使用されます。

  CP/Mにはサブディレクトリがなく、カレントディレクトリに相当するものしかないため、CP/Mプログラムはサブディレクトリの使用法を知りません。このようなプログラムがファイルをオープンすると、それはこのただ1つのディレクトリ中だけを検索します。つまり、ドライブとファイル名だけを持っていてパス名を持たないわけです。

  CP/Mプログラムが MSX DOSの下で実行されファイルをオープンしようとすると、指定されたドライブのカレントディレクトリ中でしかファイル名が検索されません。同様に、ユーザーがCP/Mプログラムにファイル名を入力する場合、ドライブとファイル名しか指定できず、つまりは、カレントディレクトリ中のファイルのみが参照されます。

  この検索がMSX-DOSを通して実行される場合、ファイルがカレントディレクトリ中で見つからないと、次にAPPEND環境変数が調べられます。それが設定されていないと、ファイルは見つからなかったとします。設定されていると、それはパス名として解釈され、ファイルの検索を続けるもうひとつのディレクトリを指定します。

  これは、CP/Mプログラムがファイルをオープンしてそれを読み書きする場合にのみ意味を持ちます。例えば、ファイルを削除あるいは作成しようとした場合にはAPPENDは使用されません。実際APPENDは望ましくない効果を生じる可能性があるためAPPENDは通常バッチファイル中だけで使用し、バッチファイル中でAPPENDをセット、CP/Mプログラムを実行、そして解除といった手順で使用するのが良いでしょう。

  APPENDの代表的な使われ方としては、大きなプログラム（ワードプロセッサやデータベースプログラム）がオーバーレイファイルやメッセージファイルを検索するディレクトリの指定、また、コンパイラ、アセンブラ、リンカがそのソースファイルやワーキングファイルを見つけるディレクトリの指定などがあります。

  APPENDが役に立たず、望ましくない効果を生む可能性がある代表的な場合としてはワードプロセッサでファイルを編集する際に、編集されたファイルは（例えAPPENDが[illegible]たぶん[illegible]た[illegible]ります。これはワードプロセッサが以下のような動作をしているからです。

  APPENDで設定したディレクトリとカレントディレクトリは異なっていて、APPENDで設定したディレクトリにあるファイル（「元のファイル」）を編集する場合、

  (1) 「元のファイル」の読み込む。

  (2) カレントディレクトリに別の名前の出力ファイルを作る。

(3) 「元のファイル」を削除しようとしたが、カレントディレクトリに「元のファイル」はないので削除でない（APPENDで設定したディレクトリに「元のファイル」は残っている)。

(4) (2)のファイルを「元のファイル」名にリネームする。

となるので、「元のファイル」がAPPENDで設定したディレクトリと、カレントディレクトリに2つできてしまいます。

- PROGRAMとPARAMETERS

これらの特殊な環境変数は外部コマンドが実行されるときにCOMMAND2.COMによって設定され、終了するときに解除されます。したがって、一般の目的で使用することは避けなければなりません。

- TEMP

パイプが実行されると(6章「リダイレクションとパイプ」を参照)、COMMAND2.COMはひとつ以上のテンポラリファイルを作成します。TEMP環境変数はこれらのテンポラリファイルが作成されるドライブとディレクトリを示します。デフォルトでは、ブートドライブのルートディレクトリが指定されていますが、一般的にはスピードを向上させるためRAMディスクを参照するように変更します。

標準のMSX-DOSシステムはパイプのためだけにTEMPを使用していますが、テンポラリファイルを作成する必要のある他のプログラムやユーティリティもTEMP環境変数を使用することができます。

- UPPER

これは外部コマンドに渡す0080h番地からのコマンド行を大文字に変換するかどうかを制御します。「ON」(小文字も可) 以外のすべての値は「OFF」として解釈されます。

UPPERが「OFF」(デフォルト) の場合、コマンド行は何の変換も行われず、タイプしたままの値が外部コマンドに渡されます。

UPPERが「ON」の場合コマンド行の文字はそれぞれ対応する大文字に変換され、外部コマンドに渡されます。これはCP/Mの環境と互換性があります。

- REDIR

これはコマンド行中のリダイレクションやパイプ文字をCOMMAND2.COMで処理するかどうかを制御します。「OFF」(小文字も可) 以外のすべての値は「ON」として解釈されます。

REDIRが「OFF」の場合リダイレクションやパイプ文字はそのまま外部コマントに渡され、外部コマンドで処理ができるようにします。

REDIRが「ON」(デフォルト) の場合、リダイレクションやパイプはCOMMAND2.COMが解釈、実行するため、外部コマンドに渡されません。

- EXPERT（2.30）

  EXPERT は DOS1 でフォーマットされたディスク上のプログラムを実行させるかさせないかを制御します。これは MSX DOS のバージョンの違いからくる問題を未然に防ぐために追加しました。「ON」(小文字も可) 以外の値はすべて「OFF」として解釈します。EXPERT が存在しない (デフォルト) か「OFF」の場合は DOS1 でフォーマットされたディスクからのプログラムの実行は禁止されます。この場合、次のようなプロンプトが出力されます。

```
*** Wrong version of MSX-DOS
```

```
*** MSX-DOS のバージョンが違います
```

  EXPERT が「ON」の場合は DOS1 でフォーマットされたディスクからのプログラムの実行が可能になります。

8.2 MSX–DOS version 2.31 の新機能

環境変数がコマンド行に取り込めるようになりました。
環境変数名はファイル名などと区別するため、前後に%をつけます。
例えば、以下のようにすると環境変数KHELP の内容を表示します。

```
A>ECHO %KHELP%
A:¥KHELP
```

8.3 環境変数の初期値一覧

環境変数は、以下のように初期設定されます。

表 2.4 環境変数の初期値

環境変数名	初期値
APPEND	なし
DATE	yy-mm-dd
ECHO	OFF
EXPERT	なし
HELP	A:¥HELP
KHELP	A:¥KHELP
PATH	:
PROGRAM と PARAMETERS	なし
PROMPT	OFF
REDIR	ON
SHELL	COMMAND.COM がロードされたところ
TEMP	A:¥
TIME	12
UPPER	OFF

9章 エラーおよびメッセージ

9.1 ディスクエラー

ディスクエラーは例えばディスクがドライブ中に入っていない場合のように、コマンドまたはプログラムがディスクにアクセスしようとし、何らかの原因で失敗した場合に起こります。エラーが発生すると、メッセージとプロンプトが表示され、(例えばディスクを正しくドライブにセットしたことなどによって) 処理を再実行する、処理を無視する、あるいはコマンド全体を中止するという選択がユーザーに与えられます。

ディスクエラーのメッセージとプロンプトの例は以下のようなもので、ドライブAがアクセスされている間にディスクが取り出された場合に表示されます。

```
Not ready reading drive A:
Abort, Retry or Ignore (A/R/I)?
```

```
ディスクが入っていません (読み込み中) ドライブは A:
中止 (A)/再試行 (R)/無視 (I)?
```

上記の例で「Notready」「ディスクが入っていません」の部分はディスクの操作が失敗した理由を示し、状況により別のメッセージ (下記参照) になります。「reading」「(読み込み中)」はコマンドがディスクを読み込み中の場合で、書き込み中の場合には「writing」「(書き込み中)」に代わります。「drive A:」「ドライブは A:」はアクセスしようとしていたドライブ名です。

「Abort,RetryorIgnore (A/R/I)?」「中止 (A)/再試行 (R)/無視 (I)?」の部分はユーザーが行うことのできる動作を示し、これらは [A]、[R]、あるいは [I] キーを押すことによって選択されます。

Abort (中止) を選択するとコマンド全体が中止され、別のコマンドを入力できるようになる前に「Disk operation aborted」「ディスク入出力が打ち切られました」というメッセージが出力されます。

Retry（再試行）を選択すると失敗したディスク操作がそのまま再実行されます。その前に、入っていなかったディスクを挿入するなどといったなんらかの動作が取られた場合には、処理は正常に行なわれます。

Ignore（無視）を選択すると、失敗したディスク操作が無視されます。多くの場合、エラーを無視するのは望ましいことではないので、その時にはIgnoreオプションは画面には表示されませんが、選択することは可能です。

エラーを無視すると、重大なシステムエラーを引き起こし、ディスク上のデータが破壊されることがあります。Ignoreオプションが表示されたとしても、他のすべての操作が失敗した場合にのみ慎重に使用して下さい。

通常、ディスク上のデータに障害が発生し、ディスクエラーを無視することがデータの全部あるいは一部を復活する唯一の可能性である場合にのみIgnoreを使用します。

重大なエラー（ディスクが使用不可能なくらい損傷しているなど）が発生した場合、処理は自動的に中止され、適当なエラーメッセージが表示されます（「Badfile allocationtable」「FAT異常です」など）。

ディスクエラーとして起こり得るエラーとその意味を以下に示します。なお、「」内は、スクリーンモードが漢字モードに設定されている場合に出力される日本語メッセージです。

表2.5 ディスクエラー一覧

英語エラーメッセージ	日本語エラーメッセージ
Bad file allocation table	**FAT異常です**

ディスクのファイルアロケーションテーブル（FAT）が異常です。

FATとは、それぞれのファイルのデータがディスク上のどこにあるのかを知らせる情報をシステムが保持するためのディスク上の領域です。したがってFATの内容が正しくないと、すべてのデータがまったく読めなくなります。このメッセージは通常、ディスクが使用できないくらい損傷していることを意味します。

Cannot format this drive **このドライブはフォーマットできません**

ディスクのフォーマッティングをサポートしていないドライブ中のディスクをフォーマットしようとしました。これは、RAMディスクを指定してFORMATコマンドを実行した場合などに発生します。

Data error **ディスクのデータが異常です**

データがエラーなしで読まれた、あるいは書き込まれたが、CRCチェックでデータのエラーが発見されました。これは通常、ディスクの障害を意味します。

Disk error **ディスクが異常です**

データがディスクに対して読み書きできませんでした。

英語エラーメッセージ	日本語エラーメッセージ

Incompatible disk **このディスクは使用できません**

2D あるいは1D のディスクをアクセスしようとしました。あるいは片面ドライブで両面ディスクをアクセスしようとしました。

Not a DOS disk **MSX-DOS ディスクではありません**

MSX-DOS が読み出せるディスクフォーマットではありません。

例えば、MSX DOS は CP/M プログラムを実行することができますが、それは CP/M のディスクを直接アクセスできることを意味するわけではありません。

Not ready **ディスクが入っていません**

ディスクがアクセスされているドライブ中に入っていません。ディスクをドライブに挿入し、「Retry」を選択します。

Sector not found **セクターが見つかりません**

MSX-[illegible]が存在[illegible]読むか[illegible]た。ディスクが損傷している可能性があります。

Seek error **シークエラーです**

ディスク上の要求されたトラックが見つかりませんでした。ディスクが損傷しているか、ディスクドライブの障害が考えられます。

Unformatted disk **ディスクがフォーマットされていません**

ディスクがフォーマットされていません。ディスクをフォーマットしてから使用して下さい。

Verify error **正しく書き込まれませんでした**

ベリファイがオンの場合にだけ起こり、データがディスクに書き込まれたように見えたが、読み出してみると、書き込んだデータと相違が発見されたことを意味します。

Write error **書き込み異常です**

データが正しく書き込まれませんでした。通常ディスクドライブの障害を意味します。

Writeprotected disk **ディスクが書き込み保護されています**

ディスクが書き込み保護されているのに、それにデータを書き込もうとしました。ディスクのプロテクトを外し、「Retry」を選択します。

英語エラーメッセージ	日本語エラーメッセージ
Wrong disk	**ディスクが違います**
Wrong disk for file	**このファイル用のディスクではありません**

MSX DOSが1度ディスクにアクセスし、その後もう1度同じディスクにアクセスする必要が生じましたが、ドライブに異なったディスクが入っていました。正しいディスクを入れて、「Retry」を選択します。

9.2 コマンドエラー

コマンドエラーはコマンドの機能を何らかの理由で実行できない場合に起こります。

エラーがコマンド中で起こり、継続できないような場合には、適当なエラーメッセージが出力されます。

エラーメッセージは例えば次のように表示されます。

```
*** File not found
```

```
*** ファイルが見つかりません
```

3つのアスタリスク「***」が最初に表示されてエラーが起こったことを示します。次にメッセージが出力され、次の行に通常のコマンドプロンプトが続きます。起こり得るすべてのエラーは後述します。

9.2.1 エラータイプ付きのエラーメッセージ

コマンドのエラーが特定の状況で起こると、「エラータイプ」メッセージも出力されることがあります。例えば、通常必要なファイルがディスク上で見つからないと、上記の例のように、「File not found」「ファイルが見つかりません」メッセージが出力されます。

必要なファイルがリダイレクション記号「<」(6章「リダイレクションとパイプ」を参照)によって指定されていると、出力されるメッセージは次のようになります。

```
*** Redirection error: File not found
```

```
*** リダイレクトエラー:ファイルが見つかりません
```

起こり得るエラータイプを以下に示します。

表 2.6 エラータイプ一覧

英語エラーメッセージ	日本語エラーメッセージ
Batch file error:	**バッチファイルエラー:**

バッチファイルから読もうとしている間に起こったエラーです。

例えば、ディスクエラーが起こって「abort」が選択されたときに出力されます。

Pipingerror:	**パイプエラー:**

パイプ処理中に起こったエラーで、多くの場合はCOMMAND2.COMが作成するテンポラリファイルとの関連で起こります（6章「リダイレクションとパイプ」を参照)。

例えば、TEMP環境変数（8章「環境変数の設定」を参照）が正しいドライブやディレクトリを参照していないときに出力されます。

Redirection error:	**リダイレクトエラー:**

リダイレクト処理中に起こったエラーです。

例えば、リダイレクション記号「<」、「>」、あるいは「>>」(6章「リダイレクションとパイプ」を参照）の後に不正なファイル名が指定されている、あるいは指定の入力ファイルが見つからないときに出力されます。

Standard inputerror:	**標準入力エラー:**

[illegible]レクションあるいはパイプがセットアップされた後に、コマンドまたはプログラムへの標準入力で起こったエラーです。

例えば、ファイルから標準入力がリダイレクトされているが、エンドオブファイルに達したときに出力されます。

Standardoutput error:	**標準出力エラー:**

リダイレクションあるいはパイプがセットアップされた後にコマンドまたはプログラムからの標準出力で起こったエラーです。

例えば標準出力がファイルへリダイレクトされているが、ディスクが一杯のときに出力されます。

9.2.2 ファイル名付きのエラーメッセージ

多くのコマンドは複数のファイルやディレクトリを操作し、ワイルドカードでファイル名を与えるとコマンドはいくつものファイルあるいはディレクトリを処理します（例えばRENAMEコマンドやCOPYコマンドなど)。コマンドを実行しているときに、あるファイルでエラーが起こったが、別のファイルでは成功すると思われる場合があります（例えば「読み出し専用」にセットせれている場合)。その場合には、ファイル名が表示され、その後にエラーメッセージが出力されてコマンドは継続します。

これは例えば次のようになります。

```
COMMAND2.COM -- File cannot be copied onto itself
```

```
COMMAND2.COM -- 自分自身にはコピーできません
```

9.2.3 エラーメッセージ

以下にアルファベット順で、起こり得るエラーを示します。

表 2.7 エラーメッセージ一覧

英語エラーメッセージ	日本語エラーメッセージ
Cannot concatenate destination file	**複写先ファイルは結合できません**

このエラーはCONCATで起こり、ソースファイルの指定に適合するファイル名のひとつが目的ファイルであることを意味します。

これは必ずしも誤りではないのですが、コマンド中での指定誤りの可能性を示します。

Cannot create destination file　　**ファイルを作成できません**

これはCOPYによって起こるエラーで、通常コピーしようとしているファイルの目的ファイルがもし作成されたとすると、すでに使用中のファイルを重ね書きしてしまうことを意味します。

これはソースファイルまたは、現在実行中のバッチファイルのような使用中のファイルにコピーしようとした場合に起こります。

英語エラーメッセージ	日本語エラーメッセージ

Cannot overwrite previous destination file **ファイルの重ね書きができません**

これはCOPYによって起こるエラーで、コピーしようとしているファイルの目的ファイルがもし作成されたとすると、前にコピーされたファイルの目的ファイルを重ね書きしてしまうことを意味します。

これは通常、意図する対象がディレクトリであるがその名前を間違えたことを意味します。

Cannot transfer above 64K **64Kを越える転送はできません**

これは通常コマンドからは起こりません。

Command too long **コマンドが長すぎます**

与えられたコマンドが長すぎます。

これはキーボードからコマンドを入力する場合には起こりませんが、バッチファイルからの場合には起こることがあります。コマンドの最大長は%パラメータの代入後127文字です。

Ctrl-C pressed **Ctrl-Cが押されました**

[CTRL] + [C] が押されてコマンドが中断されました。

Ctrl-STOP pressed **Ctrl-STOPが押されました**

[CTRL] + [STOP] が押されてコマンドが中断されました。

Directoryexists **ディレクトリが既にあります**

コマンドが、既存のディレクトリと同じ名前を持つファイルあるいはディレクトリをファイル上に作成しようとしました。

Directory not empty **ディレクトリが空ではありません**

RMDIR (RD) がファイルや他のディレクトリを含んでいるディレクトリを削除しようとしました。

削除されるディレクトリは空でなければなりません。実行するためには DEL コマンドでディレクトリの内容を削除してから RMDIR コマンドを使用して下さい。

Directory not found **ディレクトリが見つかりません**

ディレクトリコマンド (RNDIRなど) が指定のディレクトリを発見できませんでした。

英語エラーメッセージ	日本語エラーメッセージ

Disk full **ディスクがいっぱいです**

ディスク上にもう空き領域がありませんでした。この場合、コマンドを実行するためには不要なファイルを削除しなければなりません。

Disk operation aborted **ディスク入出力が打ち切られました**

ディスクエラーが起こり、「Abort」オプションが選ばれたため、コマンド全体を中断します。

Duplicatefilename **ファイル名が重複しています**

新しいファイル名が既存のファイル名と同一であるため、RENAME (REN) または RNDIR が名前の変更を実行できませんでした。

これはまた、移動されるファイルやディレクトリと同じ名前のファイル名が目的のディレクトリ中に存在する場合に、MOVE や MVDIR でも発生します。

End offile **ファイルの終わりです**

これは通常コマンドでは起こりません。

Environment string too long **環境変数が長過ぎます**

これは通常コマンドでは起こりません。

Error on standard input **標準入力でエラーが起きました**

これは通常コマンドからは起こらず、コマンドがキーボードから読み込まれようとしている間にエラーが起こったことを示します。

Error on standard output **標準出力でエラーが起きました**

これは通常コマンドからは起こらず、コマンドが画面に書き込もうとしている間にエラーが起こったことを示します。

File access violation **ファイルアクセス異常です**

これは通常コマンドでは起こりません。

File allocation error **ファイルの割当異常です**

これは通常コマンドでは起こりません。

File cannot be copied onto itself **自分自身にはコピーできません**

同一ファイルを同じファイルにコピーしようとしました。

Fileexists **ファイルが既にあります**

MKDIR (MD) で新しいディレクトリを作成しようとしたが、指定のディレクトリ中に同じ名前のファイルが存在しました。

英語エラーメッセージ	日本語エラーメッセージ

File for HELP not found — **HELPファイルが見つかりません**

HELPコマンドがヘルプテキストを得るためにファイルを探したが見つかりませんでした。

HELPファイルは通常、ブートディスクの¥HELPまたは¥KHELPというディレクトリ中にあります。

File handle not open — **ファイルハンドルがオープンされていません**

これは通常コマンドでは起こりません。

Fileis already in use — **ファイルが使用中です**

コマンドが現在実行中のバッチファイルのような、別の目的ですでに使用されているファイルを変更しようとしました。

File not found — **ファイルが見つかりません**

コマンドが指定のファイルを見つけられませんでした。

Internal error — **DOSが異常です**

これは通常コマンドでは起こりません。

Invalid MSX-DOS call — **無効なMSX-DOSファンクション番号です**

これは通常コマンドでは起こりません。

Invalid attributes — **無効な属性です**

ATTRIBあるいはATDIRで、不正な+または-属性が指定されました。

Invalid date — **無効な日付けです**

DATEコマンド中で入力された日付が正しい日付でない、あるいは不正な形式で入力されました。

Invalid device operation — **無効なデバイスオペレーションです**

コマンドがその機能を組み込みシステムデバイスに実行することができません。

例えば、ファイルはCONに名前を変更することはできません。

Invaliddirectorymove — **ディレクトリが移動できません**

MVDIRでディレクトリをそれ自身の下位ディレクトリ中に移動しようとしました。

英語エラーメッセージ	日本語エラーメッセージ
Invalid drive	**無効なドライブ名です**
存在しないドライブが指定されました。	
Invalid environment string	**無効な環境変数です**
環境変数の名前に不正な文字があります。 ファイル名に使用できる文字のみが、環境変数の名前に使用できます。	
Invalidfile handle	**無効なファイルハンドルです**
これは通常コマンドでは起こりません。	
Invalid filename	**不正なファイル名です**
ファイル名に不正な文字があります。 これはファイル名を指定、もしくはワイルドカードを使ってファイル名を変更しようとしたときに起こります。	
Invalid number	**無効な数値です**
コマンド中に指定された数値に、数字以外の文字があります。	
Invalid option	**無効なオプション指定です**
コマンド行で/の後に不正な文字が指定されました。	
Invalid . or .. operation	**. や .. に対しては操作できません**
コマンドはディレクトリの最初にある「.」と「..」の特殊なディレクトリに対して、その機能を実行することはできません。	
Invalid parameter	**無効なパラメータです**
コマンドに対するパラメータが、そのコマンドに対しては正しくありません。	
Invalid pathname	**無効なパス名です**
コマンド行で指定されたパス名が存在しない、あるいは文法的な誤りがあります。	
Invalid process id	**無効なプロセスIDです**
これは通常コマンドでは起こりません。	
Invalidtime	**無効な時間です**
TIME コマンドで入力された時刻が正しい値ではない、あるいは不正な形式で入力されました。	
Missingparameter	**パラメータが不足しています**
コマンドが必要とするパラメータ数より、ユーザー指定したパラメータの数が少ないです。	
No spare file handles	**ファイルハンドルが足りません**
これは通常コマンドでは起こりません。	

英語エラーメッセージ	日本語エラーメッセージ

Not enough memory　**メモリー不足です**

指定のコマンドで利用できるだけの十分なメモリがありません。

例えば、メモリに入りきらないくらい大きいプログラムを読み込もうとした、新しい環境文字列のための十分なメモリがない、などが原因として考えられます。

Not enough memory, system halted　**メモリが足りません．システムは停止しました**

MSX-DOSが起動して、処理を続行するのに十分なメモリがないことが分かった場合に出力される特殊なエラーメッセージです。

メッセージが示すようにシステムが停止したので、コンピュータをリセットしなければなりません。これは通常は起こりません。

Pathname too long　**パス名が長過ぎます**

パス名が長すぎます。

指定のパス名の長さが100文字を超えているか、あるいはルートディレクトリからファイルまでのパスの長さの合計が63文字を超えています。

RAM disk already exists　**RAM DISK(ドライブH:)は既にあります**

これは通常コマンドでは起こりません。

RAM disk does not exist　**RAM DISKがありません**

RAMDISKコマンドを使用してRAMディスクの現在の大きさを表示しようとしたが、RAMディスクが存在しません。

Read only file　**ファイルが読み出し専用です**

読出し専用のファイルを変更あるいは重ね書きしようとしました。

DIRコマンドによってこれを表示することができ、ATTRIBコマンドでそれを読出し専用でなくすることができます。

Root directory full　**ルートディレクトリがいっぱいです**

ルートディレクトリ中のファイル数が決められた最大値（通常112）に達しています。

サブディレクトリにはこの制限がありません。

System file exists　**システムファイルが既にあります**

作成すると、システムファイルの属性を持っているファイルに重ね書きしてしまうようなファイルを作成しようとしました。

システムファイルはMSX DOSでは使用されず、DIRコマンドで表示されません。また、他のいかなるコマンドからもアクセスできないため、通常はこのエラーはコマンドでは起こりません。

英語エラーメッセージ	日本語エラーメッセージ
Too many parameters	**パラメータが多すぎます**

コマンドが必要とするパラメータの数より、ユーザーの指定したパラメータの数が多いです。

Unrecognized command	**コマンドが違います**

指定のコマンドは内部コマンドでなく、PATH コマンドで設定された現在の検索パスにある COM あるいは BAT の外部コマンドでもありません。

Wrong version of command	**コマンドのバージョンが違います**

プログラム実行後、COMMAND2.COM はディスク上の COMMAND2.COM ファイルから自分自身を再ロードしようとしたが、それが同じバージョンではありませんでした。

プロンプトが表示され、COMMAND2.COMは再び自分自身を再ロードしようとします。

Wrong version of MSX−DOS, system halted	**MSX−DOS のバージョンが違います システムは停止しました**

MSX−DOSが起動しようとして、MSX−DOS システムの別の部分が必要なものよりも以前のバージョン番号を持つことが分かった場合に表示される特殊なエラーメッセージです。

メッセージが示すようにシステムが停止したので、コンピュータをリセットしなければなりません。これは通常は起こりません。

内部的には、エラーはエラー番号によって表現されています。そのためのメッセージがないエラー番号が受け取られるとその番号が表示されます。64以上の番号は MSX DOS の将来のバージョンのために確保されており、「システムエラー」と呼ばれます。63以下の番号は外部のアプリケーションプログラムで使用することができ、「ユーザエラー」と呼ばれます。32以下のユーザエラーはメッセージを表示しません。デフォルトのエラーメッセージ（これらは通常はコマンドでは起きない）は次のようになります。

System error 64 「システムエラー 64」

および

User error 63 「ユーザエラー 63」

ここで 64 と 63 はエラー番号の例です。エラー番号を使用する唯一のコマンドは EXIT コマンドです。前述のメッセージに対する実際の番号のリストは16章「エラー」にあります。

9.3 プロンプトメッセージ

ディスクを挿入するなどといった、システムが処理を続行する前にユーザーの動作が必要な状況がいくつかあります。また、潜在的に危険なコマンドでは処理を実行する前にプロンプトを出して確認を待つ必要があります。そのような場合に表示される種々のシステムプロンプトを以下に示します。

```
All data on drive A: will be destroyed
Press any key to continue...
```

```
ドライブB:上の全てのデータは消去されます
何かキーを押して下さい.
```

このプロンプトはFORMATコマンドで出力され、間違ったディスクを誤ってフォーマットする危険を減らします。FORMATコマンドを中止するには、[CTRL]+[STOP]または[CTRL]+[C]を押します。

```
Destroy all data on RAM disk (Y/N)?
```

```
RAMディスク上の全てのデータを消去しますか (Y/N)?
```

RAMDISKコマンドを指定してRAMディスクをセットアップしたが、RAMディスクがす[illegible]への応答が[Y][illegible]、この既存のRAMディスク上のすべてのファイルが消去されます。[N]や[CTRL]+[STOP]、[CTRL]+[C]を押すとコマンドを中止します。

```
Disk in drive A: will only be able to boot MSX-DOS
Press any key to continue...
```

```
ドライブB:のディスクはMSX-DOSしか立ち上げることが出来なくなります
何かキーを押して下さい.
```

このプロンプトは外部コマンドのFIXDISKによって出力され、MSX-DOSのものではないディスクを誤って更新してしまう危険を少なくします。FIXDISKコマンドを中止するには[CTRL]+[STOP]または[CTRL]+[C]を押します。

```
Erase all files (Y/N)?
```

```
全てのファイルを消去しますか (Y/N)?
```

DEL（またはERA、ERASE）コマンドをディレクトリ中のすべてのファイルを指定して実行したときにこのプロンプトが出力され、多くのファイルを誤って削除してしまう危険を少なくします。

```
Insert COMMAND2.COM disk in drive A:
Press any key to continue
```

```
COMMAND2.COMの入ったディスクをドライブA:に入れて下さい
何かキーを押して下さい.
```

これはプログラムを実行し終わった後に起こる場合があり、ルートディレクトリにCOMMAND2.COMを含むディスクが指定のドライブに存在する必要があります。

ドライブ（MSX DOSが最初にブートされたドライブ）にディスクを挿入後、キーを押すと続行します。COMMAND2.COMが別な場所（RAMディスクなど）にコピーされていると、そこからCOMMAND2.COMを再ロードするようにSHELL環境変数をセットすることができます（8章「環境変数の設定」を参照）。

```
Insert batch file disk in drive A:
Press any key to continue
```

```
バッチファイルの入ったディスクをドライブA:に入れて下さい
何かキーを押して下さい.
```

これはバッチファイルの実行中に起こる場合があり、システムが次のコマンドをバッチファイルから読む必要があるのにドライブに正しいディスクがないということを示します。

指定のドライブ（バッチファイルが最初に起動されたドライブ）にディスクを挿入してキーを押すと、バッチファイルの実行が正常に続行します。

```
Press any key to continue
```

```
何かキーを押して下さい.
```

このプロンプトは通常なんらかのユーザーの動作が必要なときに出力され、普通は必要な動作を記述した別のメッセージの後に出力されます。これはまたPAUSEコマンドによって出力されます。コマンドを中止するには、[CTRL]+[STOP]または[CTRL]+[C]を押します。

```
Terminate batch file (Y/N)?
```

```
バッチ処理を中止しますか (Y/N)?
```

コマンドがバッチファイル中で実行中の場合に、MSX-DOSがコマンドを[CTRL]+[STOP]や[CTRL]+[C]を押すなどして中止するとこのプロンプトが出力されます。

応答が「Y」だと、バッチファイルの処理も中止されます。「N」の場合には 中止されたコマンドに続くコマンドからバッチファイルが続行されます。

```
*** Wrong version of MSX-DOS
```

```
*** MSX-DOS のバージョンが違います
```

環境変数EXPERT (8章参照) が存在しないか、「OFF」のときに、DOS1でフォーマットされたディスクのプログラムを実行すると、このプロンプトが出力されます。ver.2.30から環境変数EXPERTとともに追加されました。

10章 Disk BASIC version 2.0

システム立ち上げ時に[illegible](MSXDOS2.SYS および COMMAND2.COM を含む) が存在しなかったり、MSX–DOS の BASIC コマンドを実行した場合、Disk BASIC version 2.0 が立ち上がります。

Disk BASIC version 2.0 は従来の Disk BASIC version 1.0 を拡張したもので、MSX–DOS2 に対応するため、RAM ディスクと階層ディレクトリの操作命令、日本語処理のための命令が追加、あるいは拡張されています。日本語処理関係の命令に関しては 11章を参照して下さい。

10.1 この章の表記法

この章では、以下の表記法を用います。

ステートメント名

機　能	命令の内容を簡単に説明しています。
書　式	命令の書き方を示します。<>で囲まれた項目は、必要とする数値や文字のデータを示します。[] で囲まれた項目は、省略可能であることを示します。
文　例	実際に命令を使った例をあげています。
解　説	その命令についての詳しい説明と、使用するときの注意などを述べています。

10.2　ステートメントの説明

以下では Disk BASIC 2.0 で追加・変更された命令について説明します。

CALL CHDIR

機　能	カレントディレクトリを変更します。
書　式	CALL CHDIR("[d:]<パス>")
文　例	`CALL CHDIR("DIR1")`
解　説	機能は MSX-DOS の CHDIR と同じです。P.79の「CHDIR」を参照して下さい。

CALL CHDRV

機　能	デフォルトドライブを切り換えます。
書　式	CALL CHDRV("d:")
文　例	`CALL CHDRV("H:")`
解　説	機能は MSX-DOS のプロンプトでドライブ名を入力したときと同じです。

CALL MKDIR

機　能	新しいサブディレクトリを作成します。
書　式	CALL MKDIR("[d:]<パス>")
文　例	`CALL MKDIR("DIR2")`
解　説	機能は MSX-DOS の MKDIR と同じです。P.115の「MKDIR」を参照して下さい。

CALL RMDIR

機能　ひとつあるいは複数のサブディレクトリを削除します。

書式　CALL RMDIR("[d:]<パス>")

文例

```
CALL RMDIR("DIR2")
```

解説　機能は MSX DOS の RMDIR と同じです。P.133の「RMDIR」を参照して下さい。

CALL RAMDISK

機能　RAM ディスクの大きさを設定、あるいは変数に代入します。

書式　CALLRAMDISK([数値], [変数名])

ただし、数値と変数名の両方を省略することはできません。どちらかを指定して下さい。

文例

```
CALL RAMDISK(32)
CALL RAMDISK(1000, A)
```

解説　変数名を指定すると、RAMDISK の容量がこの変数に代入されます。

機能はMSX-DOSの RAMDISK と同じです。P.125の「RAMDISK」を参照して下さい。

CALL SYSTEM

機能　MSX-DOS に制御を戻します。

書式　CALL SYSTEM[("<DOS のコマンド名>")]

文例

```
CALL SYSTEM("WORK")
```

解説　MSX-DOS に制御を戻します。このときコマンドを渡して、DOS に戻ってからの処理を指定することができます。CALL SYSTEM のみの時はブートドライブのルートディレクトリにある REBOOT.BAT を（もしあれば）実行します。

FILES

機能　ディスク上のファイル名、ディレクトリ名を表示します。

書式　FILES["ファイル名"][, L]

文例

```
FILES"W*.*"
```

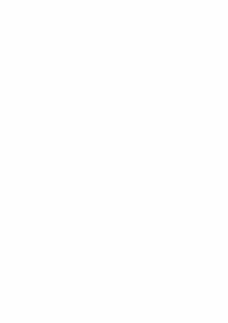
解説　機能はMSX DOSのDIRと同じです。P.96の「DIR」を参照して下さい。FILES ,Lはファイルをロングフォーマットで表示します。

10.3 Disk BASIC version 2.0 の追加エラーメッセージ

DiskBASIC 2.0 で追加されたエラーメッセージには次のようなものがあります。

表 2.8 Disk BASIC version 2.0 の追加エラーメッセージ

メッセージ	エラーコード	説明
File write protected	72	ファイルは読み出し専用である。
Directory already exists	73	ディレクトリがすでにある。
Directory not found	74	指定されたディレクトリがない。
RAMdisk already exists	75	CALL RAMDISK で RAM ディスクを作ろうとしたが、RAM ディスクがすでにある。

11章
日本語処理

1. BIOSを使用して文字を入出力している既存のプログラム（MSX DOSを含む）なら変更なしに漢字入出力可能です。

2. テキスト画面で文字にそれぞれ色指定が可能です。

3. JIS第二水準の漢字をサポートします。

4. 漢字プリンタへの漢字出力をサポートします。

5. システムに仮想端末入力インターフェイス（MSX-JE）が存在しない場合も、単漢字変換機能によって漢字の入力が可能です。

6. グラフィック画面に対し、LOCATE文、PRINT文などで、文字出力が可能です。

11.1　この章の表記法

この章では、以下の表記法を用います。

ステートメント名

機　能	命令の内容を簡単に説明しています。
書　式	命令の書き方を示します。<>で囲まれた項目は、必要とする数値や文字のデータを示します。[]で囲まれた項目は、省略可能であることを示します。
文　例	実際に命令を使った例をあげています。
解　説	その命令についての詳しい説明と、使用するときの注意などを述べています。

11.2 MSX漢字ドライバ拡張BASICステートメント

以下ではMSX漢字ドライバの拡張BASICステートメントについて説明します。

CALL KANJI

機能 漢字ドライバを起動します。

書式 CALL KANJI[n]

文例

```
CALL KANJI
```

```
CALL KANJI0
```

```
CALL KANJI1
```

```
CALL KANJI2
```

```
CALL KANJI3
```

解説 nは、0、1、2、3のうちどれか1文字で、漢字フォントとスクリーンのインタレースモードを指定します。省略すると0と見なされます。

CALL KANJI（またはCALL KANJI0）はフォントをMSXの標準漢字ROMからとり表示します。文字サイズは全角で横16ドット×縦16ドットです。

CALL KANJI1はフォントをMSXの標準漢字ROMからとり、横16ドットを12ドットに圧縮して表示します。ただし、松下電器の仕様による12ドットフォントROMがシステムに存在する場合はこのROMからフォントをとり表示します。文字サイズは12×16です。

CALL KANJI2はCALL KANJI（またはCALLKANJI0）と同じですが、インタレースモードが使用され縦方向の表示文字数が増加します。

CALL KANJI3はCALL KANJI1と同じですが、インタレースモードが使用され縦方向の表示文字数が増加します。

日本語FP は、このステートメントの実行時またはシステムの立ち上げ時に組み込まれます。

MSX が立ち上がってから一番最初の CALL KANJI の実行時には注意が必要です。その時点での HIMEM (CLEAR 文による) の設定はキャンセルされ、すべての変数およびソフトウェアスタック (FOR•NEXT GOSUB RETURN 用) がクリアされます。これは漢字ドライバのワークエリア (もしあれば日本語FP も) を常駐させる処理が行われるからです。2 回目以降の実行は問題なく行われます。

尚、本章中では、CALL KANJI 命令などを実行し漢字表示ができる状態を「漢字モード」と呼びます。さらに漢字モードでかつ SCREEN ステートメントで 0 もしくは 1 が設定されている状態を「漢字テキストモード」と呼び、漢字モードでかつ SCREEN ステートメントで 2 以上の画面モードが設定されている状態を「漢字グラフィックモード」と呼びます。

BASIC のコマンド待ちの状態では漢字テキストモードになっており、漢字の入出力が可能です。漢字グラフィックモードではプログラムによる漢字の出力のみが可能です。

漢字モードにおける WIDTH

- 漢字テキストモード

  CALLKANJI 実行後のWIDTH は ANK モードでのWIDTH をもとに、以下のように決定されます。

  - ANK モードで SCREEN 0 を使用していた場合。
    - KANJI0 か KANJI2

      ANK モードでの WIDTH の値の$\frac{1}{5}$が漢字テキストモードでの WIDTH になります。
    - KANJI1 か KANJI3

      ANK モードでの WIDTH の値がそのまま漢字テキストモードでの WIDTH になります。
  - ANK モードで SCREEN 1 を使用していた場合。
    - KANJI0 か KANJI2

      ANK モードでの WIDTHの値がそのまま漢字テキストモードでの WIDTH になります。
    - KANJI1 か KANJI3

      ANK モードでの WIDTH の値の$\frac{5}{4}$が漢字テキストモードでの WIDTH になります。

ただし、計算によって決まったWIDTHが26未満の時は26に設定されます。

- 漢字グラフィックモード

  常に表示できる最大に設定されます。

WIDTHによるSCREENモードの選択

- KANJI0かKANJI2

  WIDTHが26〜32の時は256ドットモード（VDPのモードで言うとSCREEN 5）が、33〜64の時は512ドットモード（VDPのモードで言うとSCREEN 7）が選択されます。

- KANJI1かKANJI3

  WIDTHが26〜40の時は256ドットモード（VDPのモードで言うとSCREEN 5）が、41〜80の時は512ドットモード（VDPのモードで言うとSCREEN 7）が選択されます。

漢字入力

漢字ドライバは仮想端末入力インターフェイスを持つカートリッジが存在する場合は、起動時にこれをインストールします。直接入力モード（ANK）と間接入力モード（漢字）は、[CTRL]+[SPACE] もしくは [GRAPH]+[SELECT] によって切り換えられます。仮想端末入力インターフェイスが存在しないときは単漢字変換機能（11.4「単漢字変換機能」を参照）が使用できます。

CALL ANK

機能　漢字ドライバを終了します。

書式　CALL ANK

文例

```
CALL ANK
```

解説　漢字ドライバを終了します。ただし、漢字ドライバ用に確保したメモリは開放されません。

CALL AKCNV

機能　文字を全角文字に変換します。

書式　CALL AKCNV(<文字変数>, <文字列>)

文例

```
CALL AKCNV(A$, " ABC 漢字イロハ*"):PRINT A$
ＡＢＣ漢字イロハ
Ok
```

*ゴシック体は半角文字を表わします。

解説　<文字列>中のすべての文字を全角文字に変換して<文字変数>に代入します。

CALL JIS

機能　文字を 16 進 4 桁の JIS コードに変換します。

書式　CALL JIS(<文字変数>, <文字列>)

文例

```
CALL JIS(A$, "漢字"):PRINT A$
3441
Ok
```

解説　<文字列>の最初の 2 バイトを 16 進 4 桁の JIS コードに変換して<文字変数>に代入します。

CALL SJIS

機能　文字を 16 進 4 桁のシフト JIS コードに変換します。

書式　CALL SJIS(<文字変数>, <文字列>)

文例

```
CALL SJIS(A$, "漢字"):PRINT A$
8ABF
Ok
```

解説　<文字列>の最初の 2 バイトを 16 進 4 桁のシフト JIS コードに変換して<文字変数>に代入します。

CALL KACNV

機能　文字を半角文字に変換します。

書式　CALL KACNV(<文字変数>, <文字列>)

文例

```
CALL KACNV(A$  "私はプログラマです"):PRINT A$
私ハプログラマデス
Ok
```

解説　<文字列>中の半角文字に変換できる文字をすべて半角文字に変換して<文字変数>に代入します。

CALL KEXT

機能　文字列から半角文字だけ、もしくは全角文字だけを抜き出します。

書式　CALLKEXT(<文字変数>, <文字列>, <機能>)

文例

```
CALL KEXT(A$, "今日ハ良イ天気デス ª",0):PRINT A$
ハイデス
Ok
CALL KEXT(A$  "今日ハ良イ天気デス"  1):PRINT A$
今日良天気
Ok
```

ª ゴシック体は半角文字を表わします。

解説　<機能>が0なら<文字列>中の半角文字だけを、1なら全角文字だけを抜き出し<文字変数>に代入します。

CALL KINSTR

機能　文字列中から、指定した文字列を探します。

書式　CALL KINSTR(<数値変数>[, <数式>], <文字列1>, <文字列2>)

文例

```
CALL KINSTR(A, "A亜B", "B"):PRINT A
3
Ok
```

解説　<文字列1>の中から<文字列2>を捜し出し、見つかれば発見した位置を、見つからなければ0を<数値変数>に代入します。<数式>は探し始める位置を文字数を単位として指定するもので省略されると1とみなされます。

CALL KLEN

機能　文字列の総文字数を計算します。

書式　CALLKLEN(<数値変数>, <文字列>[, <機能>])

文例

```
CALL KLEN(A, "今日ハ良イ天気です"):PRINT A
 9
Ok
CALL KLEN(A, "今日ハ良イ天気です", 1):PRINT A
 2
Ok
CALL KLEN(A, "今日ハ良イ天気です", 2):PRINT A
 7
Ok
```

解説　機能が0（もしくは省略）の時は<文字列>の全体の長さを、1の時は<文字列>中の半角文字の長さを、2の時は<文字列>中の全角文字の長さを<数値変数>に代入します。

CALL KMID

機能 文字列中から、指定した文字列を取り出します。

書式 CALLKMID(<文字変数>. <文字列>. <数式 1>[, <数式 2>])

文例

```
CALL KMID(A$, "今日はヨい天気です", 3, 3):PRINT A$
はヨい
Ok
```

解説 <文字列>中の<数式 1>番目の文字から<数式 2>文字分だけ抜きだして<文字変数>に代入します。<数式2>が省略された場合は<数式 1>番目の文字から終わりまですべての文字が代入されます。

CALL KNJ

機能 指定した漢字コードに相当する漢字を文字変数に代入します。

書式 CALL KNJ(<文字変数>, <文字列>)

文例

```
CALL KNJ(A$, "3441"):PRINT A$
漢
Ok
```

解説 <文字列>で指定される4桁の漢字コードに相当する漢字1文字を<文字変数>に代入します。漢字コードが8000H未満の時はJIS、以上の時はシフトJISとみなされます。

CALL KTYPE

機能　文字のタイプ（半角なら 0、全角なら 1）を数値変数に代入します。

書式　CALL KTYPE(<数値変数>, <文字列>, <数式>)

文例

```
10 A$="今日ハ良イ天気です"
20 CALL KLEN(L, A$)
30 FOR I=1 TO L
40 CALL KTYPE(T, A$, I):PRINT T;
50 NEXT I
RUN
 1  1  0  1  0  1  1  1  1
Ok
```

解説　<文字列>中の<数式>番目の文字のタイプ（半角なら 0、全角なら 1）を<数値変数>に代入します。

CALL CLS

機能　画面をクリアします。

書式　CALLCLS

文例

```
CALL CLS
```

解説　CALL KANJI ステートメントにより画面を漢字テキストモードにしたときに、画面をクリアするために使用します。漢字モードでないときでも使用できます。

CALL PALETTE

機　能

カラーパレットの初期化または設定を行ないます。

書　式

CALL PALETTE [(<パレット番号>, <赤輝度>, <緑輝度>, <青輝度>)]

文　例

```
CALL PALETTE ( 15, 4, 4, 4 )
Ok
```

解　説

CALL KANJI ステートメントにより画面を漢字テキストモードにしたときに、パレットを変更するために使用します。漢字モードでないときでも使用できます。

パラメータをすべて省略した場合は、パレットを初期状態にします。

パラメータが指定された場合、<パレット番号>で指定されたパレットを<赤輝度>、<緑輝度>、<青輝度>の色に設定します。なお、パレット番号や各輝度を省略することはできません。

WIDTH

機　能

1行に表示する文字数を設定します。

書　式

WIDTH <桁数>

文　例

```
WIDTH 28
```

解　説

WIDTH ステートメントは漢字モードの時には以下のように動作します。

- 漢字テキストモードの場合
  - KANJI0 か KANJI2 の場合
    <桁数>の指定は 26～64 が有効です。
  - KANJI1 か KANJI3の場合
    <桁数>の指定は 26～80 が有効です。

  上記以外の値が指定されると「Illegal function call」となります。
- 漢字グラフィックモードの場合
  常に「Illegal function call」となります。

11.3 漢字モードでの注意点

11.3.1 漢字グラフィックモードでの PRINT 文

漢字グラフィックモードでは PRINT 文によって文字出力が可能になりました。ただしこれを使用する際には以下の注意が必要です。

1. LOCATE 文で指定するカーソルポジションは、半角文字単位です。
2. カーソルポジションを保持しているエリアは1つしかないので、アクティブページを変更しながら PRINT していくと他のページでのカーソルポジションの影響を受けます。
3. インタレースモード（KANJI2、KANJI3）で文字出力をする際は、SCREEN 文で EVEN・ODD のインタレースモード（SCREEN , , , , , 3）に設定して、SET PAGE 文で表示ページを奇数ページに、アクティブページを表示ページ−1に設定しなければなりません。

11.3.2 SCREEN 文

従来 SCREEN モードを変更しただけでは、スプライト表示禁止ビット（VDP レジスタ8）やインタレースモード（VDP レジスタ9）は変化しませんでしたが、漢字モードでは両方ともクリアされます。ただし後者に関しては、SCREEN 文の第6パラメータを同時に指定した場合は、正しく設定されます。

11.4 単漢字変換機能

11.4.1 概要

単漢字変換機能は、仮想端末入力インターフェイスがないシステムであっても BASIC および MSX-DOS 上で漢字の入力を行うことができる機能で、これを用いることによって、プログラムやテキストなどで全角文字を取り扱うことができます。ただし仮想端末入力インターフェイスがある場合は、そちらが優先的に立ち上がります。

この機能は以下の特徴を持ちます。

1. 変換は漢字の音読みによって行います。
2. カーソルキーを用いて、容易に目的の漢字や特殊文字を選択す[illegible]。
3. 第二水準漢字 ROM があれば、カーソルキーによって選択・表示が可能です。
4. すべての漢字モード、スクリーンモードに対応します。

11.4.2 単漢字入力モード

CALL KANJI を実行し、漢字モードで CTRL + SPACE もしくは GRAPH + SELECT を押すことによって画面最下行が反転し、単漢字入力モードになります。この機能で、CTRL + SPACE （あるいは GRAPH + SELECT ）はトグルスイッチになっており、再びこのキーをおすことによって単漢字入力モードが終了します。

11.4.3 単漢字の入力方法

漢字の入力

前記の方法で単漢字入力モードに移行すると、仮想的な 25 行目に仮名漢字変換ウィンドウが反転表示され、単漢字変換が可能になります。

ここで「漢字」という文字を入力する場合を例に挙げます。

1. CTRL + SPACE によって漢字入力モードに入ります。画面最下行が反転表示されます。

2. かな 、もしくは SHIFT + かな （ローマ字モード）を押して、「か」を入力します。

   ```
   下化仮何伽価佳加可嘉
   ```

   （表示される文字数は最大 10 文字。文字数は漢字モード、スクリーンモードによって異なります）

   のように、「か」で始まる漢字が表示されます。

3. 続いて「ん」を入力すると

   ```
   乾侃冠寒刊勘勧巻喚堪
   ```

   「かん」で始まる漢字が表示されます。

4. 変換ウィンドウ内のカーソルを上、下、左、右または SPACE キーで移動し、目的の「漢」に合わせて、↵ キーで決定します。すると、画面に「漢」が表示され、変換ウィンドウ内はクリアされます。

5. 次に「し」、「゛」を入力（ローマ字変換の場合は、直接「じ」を入力）。

6. 同様にカーソルを「字」に合わせて ↵ キーを押します。

変換中の訂正

変換中の訂正は [ESC] キーを押すことで可能です。[ESC] キーを押すと変換ウィンドウ内がクリアされ、読み入力の状態に戻ります。

各種文字の入力方法

全角ひらがな	[かな] ランプが点灯している状態で入力します。
全角カタカナ	単漢字入力モードで [かな]、[CAPS] (CAPS LOCK) キーの両方が点灯している状態で入力します。
半角カタカナ	[CAPS] (CAPS LOCK) キーの両方が点灯している状態で入力します。
全角記号	単漢字入力モードで、[かな] ランプが点灯している状態で「を」や「ぁ」を入力すると、変換ウィンドウに記号が表示されるので、カーソルを表示したい特殊文字や記号に合わせて [↵] キーを押します。
全角英数字	単漢字入力モードで [かな] ランプが点灯していない状態で入力します。
第二水準漢字	システムに第二水準漢字 ROM がある場合は、単漢字入力モードで、[かな] ランプが点灯している状態で「ん」などを入力し、カーソルを表示したい漢字に合わせて [↵] キーを押します。

その他

- 入力された読みが存在しないときは、もっとも近い読みの漢字が表示されます。
- 目的の漢字が「音・訓」どちらの読みで登録されているかは、別表を参照して下さい。
- ローマ字変換機能については、MSX 本体のローマ字変換機能と同様です。
- 変換ウィンドウが開いているとき [CTRL] キーを押しながら連続して「0〜9、A〜F」のキーを 4 桁入力すると、それを JIS コードとして、そのコードから始まる漢字を表示します。選択方法は上に同じです。

11.5 漢字プリンタの取り扱い

MSX-DOS では、漢字を画面に出力できるだけでなく、漢字プリンタにも出力できます。出力可能なプリンタは MSX 標準の漢字プリンタとそれに準拠するものです。MSX-DOS では漢字をプリンタに出力する場合は、漢字イン ([ESC] [K]) や漢字アウト ([ESC] [H]) を必要に応じて挿入します。(漢字モード時)

MSX DOS2上から漢字を含んだファイルをプリンタに出力したい場合には次のようにします。

```
COPY KNJFILE PRN
```

また、ファイル名やディレクトリ名が漢字を含んでいる時でも、リダイレクトを利用してディレクトリをプリンタに出力することができます。

```
DIR > PRN
```

BASIC環境では、LLIST、LPRINT、LFILESコマンドによってプリンタへの漢字出力ができます。

12章 外部プログラムの環境

本章はMSX-DOSの下で外部プログラムが実行される環境について説明し、プログラムのエントリや終了、メモリの使用法などについて記述します。

12.1 MSX-DOSからのエントリ

外部プログラムが開始されるときのZ80のレジスタの内容は未定義です。RAMの0番地から始まる最初の256バイトは12.3「ページ0の使用法」で説明されるような種々のパラメータとコードでセットアップされています。

外部プログラムが開始されるときには割り込みはイネーブルで、一般的にはイネーブルのままにしておかなければなりません。MSX-DOSのファンクションコールは外部プログラムが割り込みを禁止している場合でも、割り込みを許可して戻ります。

12.2 MSX-DOSへのリターン

外部プログラムは以下の4つの方法で終了させる事ができます。

1. 呼び出し時のスタックポインタでリターン
2. 0000H番地へジャンプ
3. MSX-DOSの「プログラムの終了」ファンクションコール（P.269参照）
4. MSX-DOSの「エラーコードを返して終了」ファンクションコール（P.320参照）

1番目の方法で行うと以下のようになります。COMMAND2.COMは、外部プログラムをTPAの始めである0100H番地からロードし、スタックポインタをTPAの最後から数バイト下に設定し、0100H番地から実行します。このときスタックトップには0000Hがプッシュされています。スタックポインタをプログラム側で再設定しないでリターンすると、MSX-DOSに戻れるのはこのためです。

スタックポインタを

```
LD      sp,(0006H)
```

て再設定すると、可能な限りのRAMをスタックとして使用できますが、この場合はリターンでMSX-DOSに戻ることはできません。

スタックポインタを上記のように再設定することに不都合があるのならば、外部コマンドは自分自身のスタックをTPA中に取らなければなりません。

最初の2つはMSX DOSの下では同一で、CP/MおよびMSX-DOS1と互換性があります。3番目の方法もCP/MとMSX DOS1と互換性があり、エラーコード0の「エラーコードを返して終了」ファンクションコールを実行するのと同等です。

4番目の「エラーコードを返して終了」ファンクションを使用すると、プログラムは[illegible] DOSにエラーコードを返すことができますが、最初の3つの終了の方法は常にエラーコード0（エラ[illegible]なし）を返すだけです。

新たにMSX-DOS2で作成されるプログラムや、CP/Mプログラムでも新しくMSX-DOS2の環境に移植する場合は、エラーコードとして0を返す場合でも、4番目の「エラーコードを返して終了」ファンクションを使用して下さい。

プログラムは、その制御の範囲外で発生したイベントによっても終了することがあります。イベントとは、例えば、[CTRL]+[C]や[CTRL]+[STOP]をキーボードから入力した場合や「Abort（中止）/Retry（再試行）/Ignore（無視）」のディスクエラーメッセージに対する応答として「中止」をユーザーが選択した場合、あるいは標準I/Oチャンネルでエラーが発生した場合などです。その場合、適当なエラーコードがMSX-DOSに返されます。

外部プログラムでは「アボートルーチン」を定義することができます。

これは「プログラムの終了」あるいは「エラーコードを返して終了」ファンクションによってプログラムが終了したとき、あるいは中止エラー（上記参照）の後でコールされ、プログラムの異常終了を正しく処理することができます。このルーチンの定義の仕方およびどの場合に使用できるかについては17.3「ファンクションの説明」で説明されています。

12.3 ページ0の使用法

外部コマンド起動時に、種々のパラメータ領域がRAMの最初の256バイトに外部プログラムのためにセットアップされます。この領域のレイアウトは以下の通りで、MSX-DOS1と互換性があり、MSXのスロット切り換えルーチンのために使用される領域を除いてCP/Mと互換性があります。

0000H番地には外部プログラムを終了するために使用できるジャンプ命令があります。このジャンプの先はBIOSのジャンプベクタを見つけるために使用することもできます（12.4「BIOSジャンプテーブル」を参照）。このジャンプアドレスの下位バイトはCP/Mとの互換性のため、常に03Hです。

0003Hと0004Hの2つの予約バイトはCP/MにおけるIOBYTEとカレントドライブ番号+ユーザー番号です。MSX-DOS2はCP/Mとの互換性のために、最新のカレントドライ

アドレス	内容	
0000H	リブートエントリ　予約	MSX-DOSのエントリ
0008H	RST08H ユーザー用	RDSLTルーチンへのエントリ
0010H	RST10H ユーザー用	WRSLTルーチンへのエントリ
0018H	RST18H ユーザー用	CALSLTルーチンへのエントリ
0020H	RST20H ユーザー用	ENASLTルーチンへのエントリ
0028H	RST28H ユーザー用	予約
0030H	CALLFルーチンへのエントリ	予約
0038H	割り込みベクタ	
0040H		
0048H	拡張スロット切り換えコードによって使用	
0050H		
0058H		
0060H	最初のパラメータのためのオープンされていないCP/MのFCB	
0068H		
0070H	2番目のパラメータのためのオープンされていないCP/MのFCB	
0078H		FCBのための領域
0080H	デフォルトのディスク転送アドレス コマンド行パラメータで初期化される	
00F8H		

図2.2 ページ0のレイアウト

ブバイトを記憶していますが、MSX DOS用のプログラムではこれを参照せずに、「カレントドライブの獲得」のファンクションコールを使用して下さい。

IOBYTEとユーザー番号はI/OのリダイレクションがCP/Mと同一の方法では行われておらず、MSX-DOSにユーザー番号の概念は存在しないためサポートされていません。

0005H番地にはMSX DOSコールを実行するために使用されるMSX-DOSの常駐部分の開始番地へのジャンプ命令があります。このジャンプのアドレスは同時にプログラムが使用できるTPAの上限+1を表します。言い換えると、プログラムが使用できるのは0100H番地から、0006H番地と0007H番地で示される番地マイナス1までです。TPAのサイズはディスクインターフェイスカートリッジの種類や数によって異なりますが、一般的には約53Kです。

このジャンプのジャンプ先の下位バイトはCP/Mとの互換性のため常に06Hで、CP/Mでは、その直前の6バイトがCP/Mのバージョン番号とシリアル番号を保持しています。

0008H～0028HまでのZ80のリスタート命令の飛び先は、ユーザー用にそれぞれ4バイトが予約されていて、そこにジャンプ命令を置くことができます。ただし、リスタートの間の各々4バイトはMSXスロット切り換えルーチンへのエントリポイントとして使用されます。

0038H～005BHまでの全領域は、MSXの割り込みと拡張スロット切り換えコードのために使用されますので、ここを変更してはいけません。大部分のCP/Mのデバッガ（ZSIDやDDTなど）は38H番地をブレークポイントエントリとして使用していますので、これらのプログラムは異なったリスタートを使用するように、変更しなければなりません。RST28Hを使うことを推奨します。

005CH番地と006CH番地に設定されている2つのFCBは、コマンド行の最初の2つのパラメータをファイル名として解釈した有効なオープンされていないFCBです。どちらのファイル名も使用する場合、2番目のものは最初のものがオープンされると重ね書きされてしまうため、メモリ中の別な場所のFCBへコピーしなければなりません。FCBのフォーマットについては13.5「ファイルコントロールブロック（FCB）」を参照して下さい。

コマンド行全体（最初のコマンドを除いたもの）は0080H番地のデフォルトのディスク転送領域にストアされます。最初の1バイトには渡された文字数が入っており、最後にヌルが付加されます（ヌルと長さのバイトは、長さには含まれません）。この文字列はCP/Mとの互換性を確保するため、先頭にいくつ空白文字が入力されていてもそれを含みます。MSX DOS1との互換性を確保するため通常、文字列はそのまま（大文字に変換されずに）ストアされますが、CP/Mとの互換性を重視して環境変数UPPER（8章参照）をONに設定しておくと大文字に変換されます。

MSX-DOS用の新しいプログラムはCP/MのFCBを使用すべきではありません。なぜなら、より使うのが簡単なMSX-DOSコールが用意されていて、それを使えばディレクトリをアクセスしたりパス名の処理を行ったりすることができるからです（これらの機能についての詳細は17.3「ファンクションの説明」を参照）。

コマンド行にアクセスするのに、より改善された方法を使うこともできます。環境変数「PARAMETERS」には、コマンド行が大文字に変換されずに保持されています。別の環境変数「PROGRAM」によって、プログラムは自分がロードされたドライブ、ディレクトリ

およびファイル名を知ることができます。これらの環境変数、環境変数一般についての詳細は 13.6章を参照して下さい。

12.4 BIOS ジャンプテーブル

0000H 番地のジャンプは常にその下位バイトが 03H であるアドレスへのジャンプです。このアドレスには別のジャンプ命令があり、これは17 エントリあるジャンプテーブル中の2番目のエントリです。これは CP/M 2.2 の BIOS ジャンプテーブルに完全に対応しています。

テーブル中の最初の8つのエントリはリブートと文字入出力のためにあります。これらのルーチンは CP/M と同一の仕様でインプリメントされています。残りのジャンプは CP/M における低レベルのディスク関係のファンクションですが、MSX-DOS ではファイルシステムがまったく異なるため同等のものが存在しません。これらのルーチンはレジスタ（AF BC、DE および HL）を変更し、可能な場合にはエラーを返すことを除いて、何もせずに単にリターンします。

BIOS コールを実行中、MSX DOS は内部のスタックに切り換えるため、ユーザーのスタックは小さな領域（8バイト）しか必要ありません。

ジャンプテーブルは常に 256バイトのページ境界にありますが、TPA（0006H 番地の内容によって決まる）の上限から CP/M 2.2 で言うところの「正しい」距離にあるわけではないことに注意して下さい。これはきちんと作られた CP/Mプログラムでは決して問題になりませんが、一部のプログラムは CP/M 2.2 の BDOS のサイズに依存しているものもあるということです。それらのプログラムでは変更が必要でしょう。

BIOSのジャンプベクタ中のエントリを次に示します。

表2.9 BIOSのジャンプベクタ中のエントリ

アドレス	命令	意味
xx00H	JMP WBOOT	ウォームブート
xx03H	JMP WBOOT	ウォームブート
xx06H	JMP CONST	コンソールステータス
xx09H	JMP CONIN	コンソール入力
xx0CH	JMP CONOUT	コンソール出力
xx0FH	JMP LIST	リスト出力
xx12H	JMP PUNCH	パンチ（補助）出力
xx15H	JMP READER	リーダ（補助）入力
xx18H	JMP RETURN	CP/Mにおけるホーム
xx1BH	JMP RETURN	CP/Mにおけるディスクの選択
xx1EH	JMP RETURN	CP/Mにおけるトラックのセット
xx21H	JMP RETURN	CP/Mにおけるセクタのセット
xx24H	JMP RETURN	CP/MにおけるDMAアドレスのセット
xx27H	JMP RETURN	CP/Mにおけるセクタのリード
xx2AH	JMP RETURN	CP/Mにおけるセクタのライト
xx2DH	JMP LSTST	リストステータス
xx30H	JMP RETURN	CP/Mにおけるセクタ変換

12.5 RAMページング

外部プログラムがロードされたときには、4つのページすべてにマッパーRAMのスロットが選択され、基本の64Kを構成する4つのRAMセグメントがページングされています。MSX BIOS ROM互換のスロット処理エントリポイントがページ0で利用でき、種々のマッパーサポートルーチンがページ3で利用できます（これらの仕様については15章を参照）。

プログラムは実行中に任意にスロットの切り換えやページングを行うことができます。もちろんプログラムはページ0を変更する場合には割り込みおよびスロット切り換えのエントリポイントについて注意をしなければならず、また絶対にページ3を変更してはいけません。

MSX-DOSのファンクションコールやMSX-DOSのBIOSファンクションコールを実行する場合、任意のスロットおよびRAMセグメントをページ0、1、および2に選択でき、また、スタックも任意のページ中におくことができます。現在のページの状態はエラー状態であっても、すべてのファンクションコールで保存されます。任意のパラメータをその時選択されているページから渡すことができますが、環境変数をディスク転送だけはスロットの選

択にかかわらず、ファンクションコールが実行されるときにページングされている RAM セグメント（それらが元の TPA セグメントでなくても）に対して実行されます。

外部プログラムが TPA 以上の RAM を使用したい場合、マッパーサポートルーチン（15章参照）を使用してそれ以上の RAM を獲得できます。4つの TPA セグメント以外の RAM を使用する前に、プログラムはマッパールーチンに新しいセグメントの割り付けを命じます。これによって、新たに RAM セグメントを使用しようとするプログラムと RAM ディスクなどですでに使用中のセグメントとの競合を避けることができます。

セグメントはプログラムが終了すると自動的に解放されるように、通常「ユーザーセグメント」として割り付けて下さい。「システムセグメント」は、外部プログラムが終了した後も使用中として残しておく必要がある場合にのみ、割り付けます。

追加のセグメントを割り付ければ、プログラムはそれをページングしたり、マッパーサポートルーチンを使用してアクセスしたりできます。通常、外部プログラムは、必要な場合に再び元のページに戻せるように、元の TPA セグメントのセグメント番号を記憶する必要があります。セグメント番号は通常0、1、2、および3ですが、これらの番号を使用しているものと仮定して外部プログラムを作成してはなりません。外部プログラムは必ず別のものをページングする前に、「GET_Pn」マッパールーチンを使用してセグメント番号を知っておかなければなりません。

13章
ディスクファイルの構造

13.1　デバイスおよび文字I/O

ファイル名をMSX-DOSのファンクションに指定できる場合には、デバイス名も同じように与えることができます。これらのデバイスは文字ベースのI/Oで使用され、それによってプログラムはどちらを使用しているか知る必要なく、ディスクファイルと文字デバイスをまったく同一の方法でアクセスできるようになります。その場合にどちらを使用しているかを知る必要はありません。

デバイス名の構文はファイル名の構文と同一で、プログラムはデバイス名を扱う際に特別な処理を必要としません。このことは新しいMSX-DOS2ファンクションにもCP/M互換のFCBファンクションにも当てはまります。デバイスに使用されるため予約されているファイル名は次のとおりです。

表2.10 デバイス用予約ファイル名

デバイス名	意味
CON	画面出力・キーボード入力
PRN	プリンタ出力
LST	プリンタ出力
AUX	補助出力・入力
NUL	ヌルデバイス

これらのどれかがファイル名として指定されると、実際にデバイスが参照されます。拡張子がついている場合は無視されます。ファイルを使用する大部分のファンクションはデバイスも使用できます。例えば、CONというファイル名は「ファイル名の変更」ファンクションや「ファイルの削除」ファンクションで使用しても何の差し支えもありません。エラーは返されませんが、デバイスには何の影響も与えません。

上記のAUXデバイスはデフォルトでは何もしませんが、例えばRS-232Cドライバを利

用できるように適当なルーチンをフックすることができます。

NUL デバイスは実際には何もしません。出力文字は無視され、入力は常にエンドオブファイルとなります。

LST と PRN デバイスは同一です。

CON デバイスはキーボードからの読み込み、あるいは画面への書き込みに使用します。CON デバイスから読み出す場合、1行入力が行われ、ユーザーは行編集機能を使用できます。ユーザーが CR（リターン）を押すとはじめてその行が入力されます。入力の終わりは行頭の^Z 文字で識別されます。

システムは自動的にいくつかのファイルハンドルをこれらの標準デバイスに対してオープンします（詳細については 13.2「ファイルハンドル」を参照）。これらのファイルハンドルは、標準デバイスをアクセスするためにプログラムで使用できます。あるいは、プログラムは従来の CP/M の文字ファンクション（ファンクション 01H～0BH）を使用して、文字 I/O を行うことができます。これらの2つの方法はどちらも許されますが、それぞれ別のバッファリングの方法を使用しており、混用すると文字がバッファ中で失われてしまう可能性がありますので、通常はこれらを混用するべきではありません。

コマンド行によりリダイレクションが行われると、両方のアクセス方法（標準ファイルハンドルと文字ファンクション）がリダイレクトされます。しかし、ディスクファイルをアクセスするときには、標準ファイルハンドルを使用して大きなブロックで読み書きをする方が文字ファンクションを使用するよりもはるかに高速ですのでこちらの方が望ましいでしょう。

コマンド行によるリダイレクションが行われている場合にもプログラムではリダイレクションをされていない画面出力とキーボード入力が必要になることがあります。例えば、ディスクエラー処理ルーチンではこれが必要です。これを可能にするため、文字ファンクションのリダイレクションを一時的にキャンセルするためのファンクションが提供されています。これは 17.3「ファンクションの説明」に記述されています（P.329参照）。

13.2 ファイルハンドル

新しい MSX DOS のファンクションコールを用いてファイルのアクセスを行なう場合ファイルハンドルを使用します。また、ファイルハンドルはファイル属性の操作などのファイルの操作にも使用できます。

ファイルハンドルは特定のオープンファイルやデバイスを参照する8ビットの数値です。新しいファイルハンドルは「ファイルハンドルのオープン」(P.295参照）あるいは「ファイルハンドルの作成」ファンクション（P296参照）により割り当てられます。ファイルハンドルはファイルとのデータの読み書きに使用され、「ファイルハンドルのクロース」(P.297参照）や「ファイルハンドルの削除」ファンクション（P.309参照）がコールされるまで存在し続けます。また、ファイルの属性を変更したり名前を変更したりといった処理もファイルハンドルを使って実行することができます。

MSX-DOS が新しいファイルハンドルを割り当てるときには、使用できる最小の番号を使用します。現在のバージョンでの最大のファイルハンドルの番号は63です。将来のバージョ

ンではこれは増加する可能性がありますが、127を越えることはありません。したがってファイルハンドルが負の数になることはあり得ません。

ファイルハンドルに使用される内部のデータ構造のための領域は16KのRAM-セグメント（「データ」セグメント）中に動的に割り当てられるため、一度にオープンできるファイルの数には決まった制限がありません。このセグメントはTPAの外に置かれるため、そこにストアされるデータによってTPAのサイズが減少することはありません。システムはデータセグメント中に内部のファイルハンドル情報を保持するとともに、ディスクバッファと環境変数も保持します。

外部プログラムが実行される際には、各種のファイルハンドルが前もって定義され、オープンされています。これらのファイルハンドルは標準入出力デバイスを参照しています（13.1「デバイスおよび文字I/O」を参照）。「旧来の」CP/MスタイルのMSX-DOS文字I/Oファンクションは実際にはこれらのファイルハンドルを参照します。

外部プログラムは、コマンドインタープリタが使用していた標準入出力のファイルハンドルではなく、実際にはそのコピーを獲得します。つまりプログラムは、これらのファイルハンドルを自由にクローズしたり、別のファイルハンドルをオープンしたりでき、またこれらのファイルハンドルをプログラム終了の前に元に戻したりする必要がありません。デフォルトのファイルハンドルとその行先を次に示します。

表 2.11 デフォルトのファイルハンドル

ファイルハンドル	意味
0	標準入力 (CON)
1	標準出力 (CON)
2	標準エラー入出力 (CON)
3	標準補助入出力 (AUX)
4	標準プリンタ出力 (PRN)

コマンドインタープリタは外部プログラムなどのコマンドを実行する前に、「子プロセスの起動」ファンクション（P.318参照）を実行します。このファンクションコールはシステムに対して、新しいプログラムが「サブルーチン」として実行されようとしていることを報告します。そして、現在オープンされているファイルハンドルがすべてコピーされて、新しいプログラムはコマンドインタープリタのハンドルではなく、もとのハンドルのコピーを使用することになります。

外部プログラムが既存のファイルハンドルをクローズしたり、新しいものをオープンすることによって、なんらかのファイルハンドルを変更した場合には、それはプログラム自身のファイルハンドルのセットが変更されたことになり、もとのセットは変更されずに残ります。プログラムが終了したあと、コマンドインタープリタは「親プロセスに戻る」ファンクションコール（P.318参照）を実行し、それに最初の「子プロセスの起動」から返されたプロセスIDを渡します。これによって新しいプログラムが終了したことがシステムに報告され、その

プログラムで使用されていたすべてのファイルハンドルを捨てることができます。

それぞれのファイルハンドルについて、いくつコピーが存在するかという参照カウントが保存されているので、システムはプログラムが終了したときに必要のなくなったファイルハンドルを整理することができます。これによって、きちんと作られていないプログラムがファイルハンドルをクローズしなかったためにシステムがファイルハンドルを使いつくしてしまうような事態を防ぎます。

これらの「子プロセスの起動」と「親プロセスに戻る」ファンクションはもし有用であればユーザープログラムで使用することができます。「親プロセスに戻る」ファンクションはファイルハンドルを整理し、プログラムが解放しなかったすべてのユーザー割り当て RAM セグメントを解放します。

13.3 ディスク上のデータ構造

ファイルやサブディレクトリは、ファンクションコールを使用すれば、手軽にキメ細かく取り扱うことができます。ユーザーは、ディスク上でファイルやサブディレクトリがどのような方法で管理され、どのような形式で記録されているか、といった情報を詳しく知る必要はありません。しかし、場合によっては、ディスクの管理情報を取り出したり、それをもとにファイルやサブディレクトリとは無関係にディスクを直接アクセスしたりする手段が必要になることがあります。

MSX-DOS2 では、このような目的のために、ディスクの管理情報を得たり、「論理セクタ」を直接アクセスするファンクションコールを用意しています。

セクタを直接アクセスする場合には、どのセクタにどういった情報が書き込まれているか、という基本知識が必要になります。

13.3.1 論理セクタ

MSX-DOS2 では、3.5 インチフロッピーディスクでもハードディスクでも、あるいはその他のドライブでも基本的にはアクセスすることができます。それぞれのドライブやメディアの種類によりセクタの大きさ、トラックごとのセクタ数、記録面の数などは違っていますが、これを統一的に管理するため、MSX-DOS2 では、ディスク上の物理的な境界にとらわれず、すべてのセクタに連続した通し番号をつけて、その番号でセクタを管理する方法を採用しています。これを「論理セクタ」と呼びます。

「論理セクタ」(以下、セクタ) の番号はは 0 からそのディスクの総セクタ数−1 (ディスクの種類によって異なる) までの一連の番号によって指定します。

MSX-DOS2 では、ディスクの中のセクタを表 2.12に示す 4 つの領域に分けています。最初の 3 つの領域にはデータを管理するための情報が書き込まれ、ファイルデータの本体は「データ領域」の部分に書き込まれます。これらの位置関係は図 2.3のとおりです。ブートセクタはかならずセクタ 0 にありますが、

- FAT
- ルートディレクトリ
- データ領域の開始セクタの位置

は、メディアによって異なります。ただし、これらの情報は、ブートセクタから読み出せば得ることができます。

表 2.12 ディスクの領域

領域	内容
ブートセクタ	ディスク固有の情報と MSX-DOS2 の起動プログラム
FAT	ディスク上のファイルとサブディレクトリの位置情報
ルートディレクトリ	ディスク上のルートディレクトリの管理情報
データ領域	実際のファイルデータ

図 2.3 ディスク上の領域の位置関係

13.3.2 クラスタ

ディスクの入出力は、前述のとおりセクタが基本単位です。ただし、ファイルに対してディスク上のセクタを割り当てるときには、セクタではなく複数のセクタから成る「クラスタ」という単位が使われます。それぞれのファイルには、そのファイルサイズに応じて必要な数のクラスタが割り当てられます。1クラスタ未満の部分については、例えそれが1バイトであっても1クラスタ分のデータ領域が割り当てられます。クラスタは論理セクタと同様に連続した番号で指定されていますが、FATの項で述べる理由で、2から始まる通し番号になっており、データ領域の先頭がクラスタ#2の位置に相当します。

13.3.3　ブートセクタとDPB（ドライブパラメータブロック）

MSX-DOS2では、接続されている個々のドライブごとに「DPB」という領域がメモリ上のワークエリアに設けられ、各ドライブに固有の情報が記録されます。MSX-DOS2はどのようなタイプのディスクドライブにも対応ができますが、それはこのDPBを参照して個々のドライブに対応した処理を行うことによって、メディア間の差異が吸収できるからです。

DPBに書き込まれる情報は、ブートセクタに記録されているもので、それがMSX-DOS2の起動時やメディアが交換されるごとに変更されます。ただし、ブートセクタとDPBは、図2.4と図2.5に示すように、形式が異なっています。

オフセット	内容
00	8086のジャンプ命令
01	
02	
03	メーカーが使用する。普通、メーカー名とバージョン番号
04	
05	
06	
07	
08	
09	
0A	
0B	セクタのサイズ（バイト単位）
0C	
0D	クラスタのサイズ（セクタ単位）
0E	予約セクタ数（FAT領域の先頭セクタ）
0F	
10	FATのコピー数
11	ルートディレクトリエントリの数（作成可能なエントリの数）
12	
13	総セクタ数
14	
15	メディアID
16	FATのサイズ（セクタ単位）
17	
18	トラックあたりのセクタ数
19	
1A	ディスクの面数（片面／両面）
1B	
1C	隠されたセクタ数
1D	
1E	MSX-DOS2のブートプログラムの先頭
1F	普通+30hへのJR命令
20	「VOL_ID」という文字列
21	
22	
23	
24	
25	
26	システムが使用する • 0以外=ファイルまたはサブディレクトリを削除した後、ファイルを作っていない（UNDELが可能）。 • 0=ファイルまたはサブディレクトリを削除した後、ファイルまたはサブディレクトリを作成した（UNDELはできない）。
27	ボリュームID。値はフォーマット時に乱数で設定される。
28	それぞれの値は0〜127の値。
29	
2A	
2B	将来のため予約（0で埋めること）
2C	
2D	
2E	
2F	

図2.4 ブートセクタの情報

+0	ドライブ番号
+1	メディアID
+2	セクタサイズ
+3	
+4	ディレクトリマスク
+5	ディレクトリシフト
+6	クラスタマスク（クラスタサイズ−1）
+7	クラスタシフト
+8	FAT領域の先頭セクタ
+9	
+10	FATのコピー数
+11	ルートディレクトリエントリ数
+12	データ領域の先頭セクタ
+13	
+14	最終クラスタ番号（総クラスタ数+1）
+15	
+16	FATサイズ（セクタ単位）
+17	ルートディレクトリ領域の先頭セクタ
+18	
+19	FATバッファのアドレス（システムメモリ）
+20	

図2.5 DPBの構造

13.3.4 FAT（ファイルアロケーションテーブル）

MSX-DOS2では、1クラスタよりも大きなサイズのファイルは、複数のクラスタにまたがって記録されますが、そのとき連続した番号のクラスタが使用されるとは限りません。特に、ファイルまたはサブディレクトリの作成・削除を何度も繰り返した後では、使われなくなったクラスタがディスク上のあちこちに散在した状態になります。この状態でサイズの大きなファイルを作成すると、データは飛び飛びのクラスタに分散して置かれます。そこで「何番目のクラスタは何番目のクラスタに続いている」、というリンク情報を記録しておく場所が必要になります。それがFATの役割です。また、未使用クラスタの位置や、不良クラスタが発見されたとき以後そこをアクセスしないように記録する目的にもFATが利用されます。

FATに記録されるこのようなクラスタのリンク情報や不良クラスタ情報は、ディスクファイルを管理するうえで不可欠なものであり、一部でも破損してしまうとディスク全体が使用

できなくなる恐れがあります。そのため FAT は常に複数個のコピーが用意され、万一に備えています。

FAT の例を図 2.7に示します。FAT には、

- 先頭の 1 バイトは「FAT ID」と呼ばれ、ディスクのメディアタイプを示す値
- 次の 2 バイトはダミー値のFFH
- その次から、1 クラスタにつき 12 ビットというフォーマットで実際のリンク情報を記録している FAT エントリ

が記録されます。0 番と 1 番に相当する 3 バイトが「FAT ID」に使われていますので、ファイルのデータに対応する FAT エントリは 2 番から始まります。FAT エントリの番号は、それに対応するクラスタの番号でもあります。FAT エントリに記録された 12 ビットのリンク情報は、図 2.6のように並んでいます。リンク情報は、次に続くクラスタ番号を示す値です。もし FFFH となっているときは、そのクラスタでファイルが終了したことを意味します。

図 2.6の例では、クラスタ#2 →クラスタ#3 →クラスタ#4 という 3 クラスタ分の大きさのファイルと、クラスタ#5 →クラスタ#6 の 2 クラスタ分のファイルが存在していることがわかります。なお、クラスタが番号の小さい順にリンクしているのは図を見やすくするためで、実際には番号順であるとは限りません。

図 2.6 FAT の実例

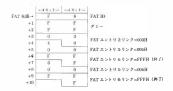



図 2.7 FATの読み方

13.3.5 ルートディレクトリ

FAT はデータの位置関係などを表すものであり、ファイル自体に関する情報は含んでいません。したがって、そのファイルの名前やそれに付随する情報を知るには、FAT とは別の情報源が必要です。これが「ディレクトリ」です。

ルートディレクトリはディスク上のルートディレクトリ領域に記録されていて、図 2.8のように 32バイトごとにディレクトリエントリ（ディレクトリの格納場所）が並んでいます。ファイルを作成すると、使われていないディレクトリエントリの中で、いちばん番号が小さいところに目的のファイルのディレクトリエントリが作られます。

ファイルまたはサブディレクトリが削除されると、該当するディレクトリエントリの最初の 1 バイトは+12「ディレクトリエントリの第 1 文字」にコピーされた後、0E5H が書き込まれ、そのディレクトリエントリが空いたことを示します。UNDEL コマンドを実行時には、この 1 バイトを元に戻してディレクトリエントリを復活させます。

ディレクトリエントリがすべて使用されてしまうと、データ領域がいくら残っていても新しいファイルまたはサブディレクトリを作ることはできません。

図 2.8 ディレクトリ領域の構造

ディレクトリエントリは図 2.9のような構造で、それぞれファイル名・ディレクトリ名・ボリューム名、属性、作成・更新の日時、先頭クラスタ番号、ファイルサイズの情報を記録しています。

ディレクトリエントリは属性によって、ファイル、ディレクトリ（ディレクトリ属性がセット）、ボリューム名（ボリューム名属性がセット）を表します。

図2.9 ディレクトリエントリの構造

属性は、ディレクトリエントリに各種の属性を与えるものです（図2.10参照）。それぞれの属性は次のような意味を持っています。

- 読み出し専用ファイル
  このビットがセットされるとファイルは書き込んだり削除したりできなくなります。読んだり、名前を変更したり、移動することはできます。
- 不可視
  このビットがセットされていると、検索属性バイト中に「不可視」ビットをセットして「最初のエントリの検索」ファンクションをコールした場合にのみ見つかります。
- システムファイル
  MSX-DOS2のファンクションが関知する限り、このビットは「新しいエントリの検索」と「作成」ファンクションコールが自動的にシステムファイルを削除することはないということを除いて、「不可視」ビットと全く同じ効果を持ちます。コマンドインタプリタによって組み込まれたコマンドでは、システムファイルをアクセスすることができません。

- ボリューム名
  このビットがセットされるとこのエントリはボリュームの名前を定義します。これはルートディレクトリでのみ可能で、ひとつだけしか設定できません。他のビットはすべて無視されます。

- アーカイブ
  ファイルが書き込まれてクローズされると必ずこのビットがセットされます。このビットは、XCOPYコマンドなどによって検出され、ファイルが変更されたかどうかを判断します。MSX DOS1ではこのビットは常に0です。

図2.10 属性

日付と時刻は図2.11と図2.12のように、それぞれ2バイトの領域を3つのビットフィールドに分割して記録しています。「年」は7ビットに0〜99の値を設定することで、西暦1980年〜2079年を表します。「秒」用のビットフィールドは5ビットで、時間の分解能は2秒です。

図2.11 日付を表すビットフィールド

図 2.12 時刻を表すビットフィールド

13.3.6 ボリューム名

ディレクトリエントリのボリューム名属性がセットされているとそのエントリはボリューム名を表します。ボリューム名はルートディレクトリでのみ設定可能で、またひとつだけしか設定できません。ボリューム名は11バイトであり、コントロールコードと「/」を除いてファイル名としては無効な文字を含めることができますが、先頭に空白は入りません。

13.3.7 サブディレクトリ

MSX-DOS2ではサブディレクトリによって階層ディレクトリを扱えるようになりました。

サブディレクトリはデータ領域内にとられ、その領域はファイルと同じようにFATで管理されます。サブディレクトリの領域にはルートディレクトリと同じようにディレクトリエントリが並んでいます。ただし、この領域の最初の2つのエントリは特別な目的に使われます。これらのエントリはディレクトリ属性を持ち、ディレクトリ名は「.」「..」になります。「.」の先頭クラスタ番号は自分自身のディレクトリを指し、「..」の先頭クラスタ番号は親ディレクトリを指します。

13.3.8 クラスタからセクタへの換算

FATやディレクトリでは、ディスク上のデータの位置はクラスタ単位で表わされています。クラスタで示されたこれらのデータをファンクションコールでアクセスするためには、あるクラスタが何番のセクタに対応しているか、という関係を求めなければなりません。これはデータ領域がクラスタ#2から開始していることを元に、以下のように計算することができます。

1. 与えられたクラスタ番号をCとする。
2. データ領域の開始セクタを調べ、これをS0とする。
3. 1クラスタが何セクタに相当するか調べ、これをnとする。
4. 求めるセクタ番号Sは、S=S0+(C−2)×nの計算で得られる。

13.4 ファイル情報ブロック（FIB）

ディスク上のファイルに作用するすべての新しい MSX-DOS ファンクションには、ヌル文字で終了する文字列（ASCIIZ 文字列と呼ぶ）への単なるポインタを渡すことができ、この文字列にはドライブ、パス名、およびワイルドカードを使用しない確定したファイル名を指定することができます。これらは通常外部プログラムが主に高級言語インタフェー[illegible]通して実行する処理です。詳細は 5.1「表記法について」を参照して下さい。

これらの ASCIIZ ファンクションには、かわりにファイル情報ブロック（FIB File Information Block）を渡すことができます。FIB は未知のファイルやサブディレクトリをディレクトリ上で検索するといった、より複雑な処理に使用されます。通常、これらはコマンドインタープリタやユーティリティのみで使用されていて、外部プログラムでしか使わないでしょう。

FIB は特定のファイルやサブディレクトリのディスク上のディレクトリエントリの情報を持っているユーザーメモリ中の 64 バイトの領域です。FIB 中の情報は新しい MSX-DOS の「検索」ファンクション（「最初のエントリの検索」(P.291)、「新しいエントリの検索」(P.293)、および「次のエントリの検索」(P.292)）によって満たされます。FIB の形式は次のとおりです。

表 2.13 FIBの形式

アドレス		内容
	0	常に 0FFH
1～	13	ファイル名（ASCIIZ文字列）
	14	ファイル属性バイト
15～	16	最終変更時刻
17～	18	最終変更日付
[illegible]～	20	開始クラスタ
21～	24	ファイルのサイズ
	25	論理ドライブ
26～	63	内部情報（変更してはならない）

どちらの形式のパラメータもとることができるファンクションがあるため、FIBの最初の「0FFH」はパス名文字列と区別するために必要です。

「ファイル名」は直接印字可能なフォーマットでストアされ、ASCIIZ 文字列の形式になります。空白はすべて取り除かれ、もしあればファイル名拡張子の前にピリオドが付加され、名前は大文字にされます。エントリがボリュームラベルであると、名前は「.」セパレータなしでストアされ、空白が残されて大文字化されません。

「ファイル属性バイト」はファイルに関するフラグから成るバイトです。このバイトのフォーマットを以下に示します。

- ビット0 – 読み出し専用
  このビットがセットされるとファイルは書き込んだり削除したりできなくなりますが読んだり、名前を変更したり、移動することはできます。

- ビット1 – 不可視ファイル
  このビットがセットされていると、「最初のエントリの検索」ファンクションを、検索属性バイト中に「不可視ファイル」ビットをセットしてコールした場合にのみ、ファイルが見つかります。

  コマンドインタープリタに組み込まれたディスク上のファイルやディレクトリにアクセスするすべてのコマンドは、「/H」オプションを指定することによって不可視ファイルを見つけることができます。

- ビット2 – システムファイル
  MSX-DOSのファンクションに関する限り、このビットは、「新しいエントリの検索」と「作成」ファンクションコールが自動的にシステムファイルを削除することはないということを除いて、「不可視ファイル」ビットとまったく同じ効果を持ちます。コマンドインタープリタによって組み込まれたコマンドでは、システムファイルをアクセスすることができません。

- ビット3 – ボリューム名
  このビットがセットされているとこのエントリはボリュームの名前です。これはルートディレクトリにひとつだけしか存在できません。他のビットはすべて無視されます。

- ビット4 – ディレクトリ
  このビットがセットされてるとこのエントリはファイルではなくサブディレクトリです。したがって読み出しや書き込みのためにオープンできません。サブディレクトリの場合は、「不可視ファイル」ビットだけが意味を持ちます。

- ビット5 – アーカイブビット
  ファイルが書き込まれてクローズされると必ずこのビットがセットされます。このビットは、XCOPYコマンドなどによって、ファイルが変更されたかどうかを判断するために、検査できます。

- ビット6 – 予約（常に0）

- ビット7 – デバイスビット
  これがセットされているとFIBがディスクファイルでなく、文字デバイス（「CON」など）を参照していることを示しています。他のすべての属性ビットは無視されます。

「最終変更時刻」は次のように2バイトにエンコードされます。

表 2.14 最終変更時刻の2バイトエンコード

ビット	意味
15~11	時間 (0~23)
10~ 3	分 (0~59)
4~ 0	秒/2 (0~29)

「最終変更日付」は次のように2バイトにエンコードされます。すべてのビットがゼロであると、日付がセットされていないことになります。

表 2.15 最終変更日付の2バイトエンコード

ビット	意味
15~ 9	年 (0~99・1980~2079 に対応)
8~ 5	月 (1~12・1月~12月に対応)
4~ 0	日 (1~31)

「ファイルサイズ」は最下位バイトが先頭にストアされた32ビットの数値で、サブディレクトリでは0です。

「論理ドライブ」は1バイトのドライブ番号で、1がA:、2がB:というように対応します。元のファンクションでゼロが指定された場合(それはカレントドライブという意味なので)カレントドライブのドライブ番号がここに入れられるため、ゼロになるということはありません。

「内部情報」によって、MSX-DOSはディスク上のどこにディレクトリエントリがストアされているかを知ります。これによってFIBを渡されたファンクションで、例えば削除、名前の変更やオープンなどの、ディレクトリエントリに対する処理ができるようになります。また、ここにストアされたデータによって、「次のエントリの検索」ファンクション(P.292参照)が次に一致するファイルの検索を実行できます。

ユーザーは内部の情報をアクセスしたり修正したりしてはなりません。

FIBは「最初のエントリの検索」、「新しいエントリの検索」、および「次のエントリの検索」MSX-DOSファンクションによって書き込まれます。これらのファンクションはディレクトリエントリを探し、適切な情報をFIBに書き込みます。

「最初のエントリの検索」の場合には、ディレクトリは指定のファイル名に一致し、適合する属性を持つ最初のエントリについて検索されます(詳細については17.3「ファンクションの説明」を参照)。「次のエントリの検索」は直前の「最初のエントリの検索」ファンクションによって始められた検索を実行し、次に一致するエントリでFIBを更新します。

「新しいエントリの検索」(P.293参照) は「最初のエントリの検索」に類似していますが、一致するエントリを検索するかわりに新しいエントリを作成し、「最初のエントリの検索」で見つかったのと同様にFIBを返します

「検索」ファンクションのいずれかでFIBを作成すると、それは2つの方法で利用することができます。

最初の方法はファイル名やサイズなどといった、それが持っている情報を単純に利用するものです。例えば、「DIR」コマンドは単純に情報を画面に出力します。

FIBを使用するもう1つの方法は、ディレクトリエントリについてのなんらかの処理を実行するために、それをもう1度別のMSX-DOS2ファンクションに渡す方法です。

17.3「ファンクションの説明」で記述されているMSX-DOS2ファンクションの多くはDEレジスタに入っているポインタを、ドライブ・パス・ファイル文字列あるいはFIBのどちらかを指してもよいものとして使用します。どちらの場合も、ファンクションには実行の対象として特定のファイルやディレクトリが指定されます。

このようなパラメータを取ることのできるファンクションは「ファイルあるいはサブディレクトリの削除」(P.304参照)、「ファイル名あるいはサブディレクトリ名の変更」(P.305参照)、「ファイルあるいはサブディレクトリの移動」(P.306参照)、「ファイルの日付および時刻の獲得・セット」(P.308参照)、および「ファイルハンドルのオープン」(P.295参照) です。これらのすべては、指定のファイルあるいはディレクトリについて、所定の機能を実行します。

FBIは「最初のエントリの検索」あるいは「新しいエントリの検索」ファンクションへもドライブ・パス・ファイル文字列の代わりに渡すことができます。この場合、FIBはファイルではなくディレクトリを参照しなければならず、ファイル名文字列をHLレジスタで渡さなければなりません（通常ヌル文字列で、これは「*.*」と同じ意味)。FIBで指定されたディレクトリでは、ファイル名とマッチするものが検索され、通常の属性のチェックを受けます。この機能はコマンドインタープリタで、UTILがディレクトリである場合に「DIR A:UTIL」などといったコマンドが必要な動作を実行するために必要です。

13.5　ファイルコントロールブロック（FCB）

MSX-DOS2の外部プログラムやMSX-DOS2用に修正されたMSX-DOS1やCP/Mのプログラムが CP/M互換のFCBファンクションを使用することは想定されていませんが、これらのファンクションで使用されるFCBのフォーマットを参考のために説明します。

このフォーマットはCP/MやMSX-DOS1で使用されるFCBに非常に似ていますが、FCB中のいくつかの領域の使用法は異なります。

基本的なFCBの長さは33バイトです。このタイプのFCBはファイル管理操作（削除、名前の変更など）や、シーケンシャルな読み書きで利用できます。ランダムな読み書きのファンクションは、ランダムレコード番号を格納するためにFCBの後から更に3バイト使用しています。MSX-DOS1互換のブロックリード・ライトのファンクションもこの追加の3バイト（場合によっては4バイト）を使用します。詳細については17.3「ファンクションの説

明」を参照して下さい。

FCBのレイアウトを以下に示します。それぞれの領域の大まかな説明もここに書いてあります。17.3「ファンクションの説明」で記述されている個々のファンクションの説明では、領域がそれぞれのファンクションでどのように使用されるかが詳しく説明されています。

- 00H　ドライブ番号（1~8。0以下ならばカレントドライブ）
  使用されるすべてのFCBでセットアップされなければならず、MSX-DOSのファンクションコール（環境変数「APPEND」が使用されている場合の「ファイルのオープン[FCB]」(P.275参照)を除く）では変更しません。

- 01H~08H　ファイル名（左詰めにして後に空白が付く）
  ファイル名にワイルドカードが使用できる場合には「?」、「*」文字を含むことができます（P.275「ファイルのオープン[FCB]」を参照）。比較を行う場合、大小文字の区別は無視されます。新しいファイルを作成する場合、名前は大文字にされます。

- 09H~0BH　ファイル名拡張子
  ファイル名と同じです。ファイル名拡張子の文字のビット7はCP/Mと異なり、フラグとして解釈されません。

- 0CH - エクステント番号（下位バイト）
  外部プログラムによってオープンや作成の前に（通常ゼロに）セットされなければなりません。

  これはシーケンシャルリード・ライトによって使用、更新され　また、ランダムリード・ライトによってセットされます。CP/MおよびMSX-DOS1と互換性があります。

- 0DH - ファイル属性
  「ファイルのオープン」(P.275参照)、「ファイルの作成」(P.280参照)と「最初のエントリの検索」(P.277参照)によってセットアップされます。

- 0EH

  CP/Mファンクションでのエクステント番号（上位バイト）
  オープンと作成によってゼロにされます。シーケンシャルリード・ライトでは、エクステント番号の拡張として使用、更新され、CP/Mでアクセス可能なものより大きなファイルをアクセスできるようにします。これはCP/Mとは異なるが、CP/M風にFCBを使用する妨げにはならず、MSX-DOS1と同じです。

  MSX-DOS1互換のブロックファンクションでのレコードサイズ（下位バイト）
  ブロックリード・ライトファンクションを使用する前に、必要なレコードサイズにセットしなければなりません。

- 0FH　CP/Mファンクションのレコードカウント
  オープンおよび作成によってセットアップされ、シーケンシャルおよびランダムリー

ド・ライトによって必要に応じて修正されます。これは CP/M および MSX-DOS1 と同じです。

MSX DOS1 互換のブロックファンクションでのレコードサイズ (上位バイト) ブロックリード・ライトファンクションを使用する前に、必要なレコードサイズにセットしなければなりません。

- 10H〜13H　バイト単位でのファイルサイズ (最下位バイトが最初)
  ファイルサイズは正確なもので、128バイト単位に切り上げられません。この領域はオープンおよび作成によってセットアップされ、ファイルが書き込み処理によって拡張されたときに更新されます。クローズファンクションコールによってディスクに書き込まれるため、外部プログラムが変更してはいけません。これは MSX-DOS1 と同じですが、ここに割り当て情報を格納する CP/M とは異なります。

- 14H〜17H - ボリューム ID
  これは、この FCB がアクセスしている特定のディスクを識別する 4 バイトの数値です。オープンおよび作成によってセットアップされ、読み出し、書き込み、およびクローズコールでチェックされます。プログラムによって変更してはいけません。これは、ここに最終更新日付および時刻をストアする MSX-DOS1、割り当て情報を格納する CP/M とは異なります。

- 18H〜1FH　内部情報
  これらのバイトはファイルをディスク上で見つけるための情報を持ちます。外部プログラムによってはいかなる変更もしてはいけません。ここに保持されている内部情報は、MSX-DOS1 によって保持されるものと似てはいるが同一ではなく、CP/M のものとはまったく異なります。

- 20H　エクステント中のカレントレコード (0〜127)
  最初のシーケンシャルリード・ライトの前に外部プログラムによって (通常ゼロに) セットされなければなりません。シーケンシャルリード・ライトによって使用、変更されます。また、ランダムリード・ライトによってセットアップされます。これは CP/M および MSX-DOS1 と互換性があります。

- 21H〜24H - ランダムレコード番号 (下位バイトが先)
  この領域はオプションで、ランダムあるいはブロックリード・ライトが使用される場合にだけ必要となります。これらの処理を実行する前にセットアップしなければならず、ブロックリード・ライトによって更新されるが、ランダムリード・ライトによっては更新されません。「ランダムレコードのセット」ファンクションによってもセットされます。

  ブロックリード・ライト (MSX-DOS1 に存在して CP/M には存在しない) では、レコードサイズが64バイト未満の場合には4バイトすべてが使用され、レコードサイズが 64 バイト以上の場合には最初の3バイトだけが使用されます。ランダムリード・ラ

イトは最初の3バイトだけが使用されます（暗黙のレコードサイズは128バイト）。これは CP/M および MSX-DOS1 と互換性があります。

13.6 環境変数

MSX DOS2 はデータセグメント中に「環境変数」のリストを持っています。環境変数とは、MSX-DOS2 やアプリケーションプログラムが動作するときに必要となる値を設定しておく変数です。環境変数には、DOS2 やアプリケーションプログラムが使用する特定のものと、ユーザーが定義できるものとがあります。名前と値はどちらもユーザーが定義します。環境変数は「環境変数の獲得」(P.328参照)、「環境変数のセット」(P.328参照) および「環境変数の検索」ファンクション (P.329参照) を通してファンクションコールレベルでアクセスすることができます。

環境変数の名前はファイル名で使用できる任意の文字で構成されるヌルでない文字列です。環境変数は255文字の長さまで許されます。環境変数は文字列が定義されるときに大文字に変換されますが、名前が比較されるときは、大小文字の区別をしません。

環境変数の値はヌルでない文字の文字列で構成され、255文字までの長さが可能です。環境変数の値がヌル文字列にセットされると、名前は環境変数のリストから除去されます。同様に、定義されていない環境変数の値が読み出されると、ヌル文字列が返されます。値は大文字にされず、値の文字列中の文字については何の変換も行われません。

外部プログラムがロードされ COMMAND2.COM から実行されると、外部プログラムが読み出すことのできる2つの特殊な環境変数がセットアップされます。

PARAMETERS という環境変数は実際のコマンド名を含まないコマンド行の内容です。これは、CP/M との互換性のため、80H にセットアップされるものと似ていますが、大文字にはされません。

PROGRAM というもうひとつの環境変数は、ディスク上のプログラムを見つけるために使用される完全なパスで、ドライブ、ルートからのパス、プログラムの実際のファイル名の順になっています。ドライブ、パス、およびファイル名は「パス名の解析」ファンクションコール (P.314参照) を使用して分離することができます。

PROGRAM 環境変数にはいくつかの使用法があります。主要な用途はプログラムがそれを、プログラムがロードされたところと同じディレクトリからオーバーレイファイルをロードするために利用できることです。PROGRAM 中の最後の項目（つまり実際のプログラムのファイル名）をオーバーレイファイルの名前に置き換えて、新しい文字列を ASCIIZ 文字列を取る任意の新しい MSX-DOS2 ファンクション（「ファイルハンドルのオープン」など）に渡すことができます。

CP/M プログラムのいくつかは外部プログラムをロードして実行することができますがこの場合 CP/M プログラムは PROGRAM や PARAMETERS 環境変数をセットアップしませんので、環境変数は CP/M プログラムがロードされたときのままになっています。

プログラムが PROGRAM および PARAMETERS を使用しようとして、しかも CP/M プログラムからもロードできるようにしようとする場合には、ページ0の 0037H 番地にある

「LOAD_FLAG」という変数を調べることができます。このフラグは、すべてのMSX-DOS2のファンクションコールでゼロにセットされ、外部プログラムがCOMMAND2.COMによって実行される直前にはゼロ以外の値にセットされます。MSX-DOS2用の外部プログラムが他の外部プログラムをロードして実行できる場合、PROGRAMとPARAMETERSをセットアップするには、COMMAND2.COMと同様にLOAD_FLAGをゼロ以外にセットしなければなりません。

もうひとつの特殊な環境変数はAPPENDです。これはユーザーがコマンドインタープリタからセットアップし、CP/Mの「ファイルのオープン(FCB)」ファンクション(P.275参照)で使用されます。このファンクションコールが実行されてファイルが見つからないと、APPENDで指定される別のディレクトリが検索されます。ただし、MSX DOS2用の外部プログラムがこのファンクションコールやAPPEND環境変数を使用するということは考えられていません。

種々のシステムの機能やオプションを制御するため、いくつかの環境変数がコマンドインタープリタが起動するときにセットアップされたり、ユーザーによって変更されたりするので、外部プログラムでそのいくつかを読み込むと便利です。例えば、PATH環境変数や、プログラムが日付や時刻を出力する場合にはDATEやTIME環境変数を読み込むと便利です。8章「環境変数の設定」ではこれらのデフォルトの環境変数の詳細を記述しています。

14章 画面制御コード

以下に示すのは、MSX-DOS2の文字ファンクションによる文字の出力やBIOSコールを実行するとき、あるいはCONデバイスへ書くときに使用することができるすべての制御コードとエスケープシーケンスのリストです。これらはMSX-DOS1と互換性があり、VT-52制御コードを含みます。画面は2～80文字×24行です。印字可能文字が表示されるとカーソルは次の位置に移動し、行の終わりの場合には次の行の先頭に移動します。文字が画面の右下に書かれると、画面はスクロールして次の行の最初にカーソルを置きます。エスケープシーケンス中の文字は大小文字の区別を正しく行わねばなりません。また、読みやすいように空白が入れてありますが、空白はシーケンスの一部ではありません。数値（<n>や<m>で示されている）は通常20Hのオフセットを加えた単一バイトとしてシーケンス中に含まれます。

表 2.16 制御コード

シーケンス	コード	機 能
CTRL+G	07H	ベル
CTRL+H	08H	カーソルを左に。前の行にラップアラウンドして画面の左上で停止。
CTRL+I	09H	タブ。次の8番目のカラムまで空白で埋め、次の行の始めにラップアラウンドし、画面の右下でスクロール。
CTRL+J	0AH	改行。画面の最下行ではスクロール。
CTRL+K	0BH	カーソルをホームポジションへ。
CTRL+L	0CH	画面をクリアしてカーソルをホームポジションへ。
CTRL+M	0DH	復帰。
CTRL+[	1BH	エスケープ(後のエスケープシーケンスを参照)。
CTRL+¥	1CH	カーソルを右に。次の行にラップアラウンドして画面の右下で停止。
CTRL+]	1DH	カーソルを左に。前の行にラップアラウンドして画面の左上で停止。
CTRL+^	1EH	カーソルを上に。画面の一番上で停止。
CTRL+_	1FH	カーソルを下に。画面の一番下で停止。
DEL	7FH	文字を削除してカーソルを左に移動。前の行にラップアラウンドして画面の一番上で停止。

表 2.17 エスケープシーケンス

シーケンス	コード	機 能
ESC A	1BH 41H	カーソルを上に。画面の一番上で停止。
ESC B	1BH 42H	カーソルを下に。画面の一番下で停止。
ESC C	1BH 43H	カーソルを右に。行の終わりで停止。
ESC D	1BH 44H	カーソルを左に。行の始めで停止。
ESC E	1BH 45H	画面をクリアしてカーソルをホームポジションに移動する。
ESC H	1BH 48H	カーソルをホームポジションに移動する。
ESC J	1BH 4AH	カーソルはそのままで、画面の一番下まで消去する。
ESC j	1BH 6AH	画面をクリアしてカーソルをホームポジションに移動する。
ESC K	1BH 4BH	カーソルはそ[illegible]ままで、行の終わりまで消去する。
ESC L	1BH 4CH	カーソル行の[illegible]に行を挿入し、画面の残りの部分を下[illegible]スクロールする。カーソルは新しい空[illegible]の始めに置く。
ESC l	1BH 6CH	行全体を消去[illegible]る。カーソルはそのまま。
ESC M	1BH 4DH	カーソル行を[illegible]除し、画面の残りの部分を上にスクロ[illegible]ルする。カーソルは次の行の最初に置[illegible]。
ESC x 4	1BH 78H 34H	ブロックカー[illegible]ルを選択する。
ESC x 5	1BH 78H 35H	カーソルを表[illegible]しない。
ESC Y <n><m>	1BH 59H <n> <m>	カーソルを行[illegible]n>列<m>に置く。画面の左上隅[illegible]n=m=20H（空白）。
ESC y 4	1BH 79H 34H	カーソルの形[illegible]をアンダーラインにする。
ESC y 5	1BH 79H 35H	カーソルを表示する。

15章 マッパーサポートルーチン

MSX-DOS2にはメモリマッパーのサポートを提供するルーチンが含まれています。これによってMSXのアプリケーションプログラムやMSX-DOSの外部プログラムは、RAMディスクや他のすべてのシステムソフトウェアと衝突することなしに、基本の64Kのメモリよりも多くのメモリを使用することができるようになります。

15.1 マッパーの初期化

MSXマシンでは、DOSのカーネルは初期化される際、システムにメモリマッパーが存在することをチェックし、メモリマッパー上に最低128KのRAMが存在していることを確認します。最低128KのマッパーRAMが存在するスロットが1つでも見つかるとカーネルは、それらのうち最大のRAM容量を持つスロット（同じ容量のマッパースロットがある場合、[illegible]番号の小[illegible]Mとして使用できるようにします。このスロットをプライマリマッパースロットといいます。メモリマッパー上に十分なメモリがないと、MSX-DOS2は立ち上がりません。

MSX turbo Rマシンでは、常にスロット３０がプライマリマッパースロットになります。

次にカーネルは、プライマリマッパースロットで利用できるすべての16KのRAMセグメントのテーブルを作成します。ユーザー用の64Kを構成する最初の４つと最も大きい番号の付けられた２つのセグメントはシステム用に割り当てられ、そのひとつはDOSのカーネルコードに、もうひとつはDOSのカーネルの作業領域として割り当てられます。他の（最低２つ）セグメントは初期状態では未使用として登録されます。さらにカーネルは（もしあれば）その他のマッパーRAMスロットについて、同様のテーブルを作成します。これらのセグメントは初期状態ではすべて未使用として登録されます。

15.2 マッパー変数とルーチン

マッパーサポートルーチンはMSX-DOSのシステムエリアにあるいくつかのテーブルを更新します。これらのテーブルはユーザープログラムから参照して種々の目的に利用するこ

とはできますが、絶対に変更してはなりません。テーブルの内容を以下に示します。

表 2.18 マッパーサポートルーチンが更新するテーブル

アドレス	機能
+0	マッパースロットのスロットアドレス。
+1	16KRAM セグメントの総数。1~255（プライマリでは 8~255)。
+2	未使用の 16KRAM セグメントの数。
+3	システムに割り当てられた 16KRAM セグメントの数（プライマリでは最低 6)。
+4	ユーザーに割り当てられた 16KRAM セグメントの数。
+5~+7	システム予約。常にゼロ。
+8~	他のマッパースロット（複数）のエントリ。ない場合+8 はゼロ。

プログラムは種々のサブルーチンを呼ぶことによってマッパーサポートルーチンを使用します。これらのルーチンは、MSX DOS のシステムエリアにあるジャンプテーブルからアクセスすることができます。ジャンプテーブルの内容は以下のとおりです。

表 2.19 ジャンプテーブルの内容

アドレス	エントリ名	機能
+0H	ALL_SEG	16K のセグメントを割り当てる。
+3H	FRE_SEG	16K のセグメントを解放する。
+6H	RD_SEG	アドレス A:HL から A にバイトを読む。
+9H	WR_SEG	E からアドレス A:HL へバイトを書く。
+CH	CAL_SEG	セグメント間コール。IY:IX のアドレス。
+FH	CALLS	セグメント間コール。コール命令の後の行のアドレス。
+12H	PUT_PH	セグメントをページ (HL) に置く。
+15H	GET_PH	ページ (HL) の現在のセグメントを得る。
+18H	PUT_P0	セグメントをページ 0 に置く。
+1BH	GET_P0	ページ 0 の現在のセグメントを得る。
+1EH	PUT_P1	セグメントをページ 1 に置く。
+21H	GET_P1	ページ 1 の現在のセグメントを得る。
+24H	PUT_P2	セグメントをページ 2 に置く。
+27H	GET_P2	ページ 2 の現在のセグメントを得る。
+2AH	PUT_P3	ページ 3 は絶対に変更してはならないため、サポートされていない。コールされると「NOP」のように動作する。
+2DH	GET_P3	ページ 3 の現在のセグメントを得る。

プログラムでこれらのアドレスを得るには、マッパーサポートの拡張 BIOS コールを使用します。これは将来のバージョンでアドレスが変更される、あるいは MSX-DOS2以外のマッパーサポートルーチンを使用できるようにするための処置です。

拡張BIOS を使用するには以下のようにします。ますプログラムはページ3の FB20h にある「HOKVLD」のフラグを調べます。このバイトのビット 0（LSB）が 0 なら、拡張 BIOS は存在せず、マッパーサポートもありません。ここが 1 の場合、以下に述べる「EXTBIO」のエントリはセットアップされており、種々のパラメータを持ってコールすることができます。ただし、MSX-DOS が存在することが確実なアプリケーション（例えばディスクからロードされるプログラム）ではこのチェックは不要で、直ちに次のステップに進むことができます。

次にプログラムは、D レジスタに拡張 BIOSのデバイス番号、E レジスタにファンクション番号を入れ、必要なパラメータを他のレジスタに入れ、ページ 3 の 0FFCAH にある「EXTBIO」をコールします。この際スタックポインタはページ 3 になければなりません。指定したデバイス番号の拡張 BIOS が存在する場合、レジスタ（AF、BC および HL）はファンクションに応じて変更され、存在しない場合、保存されます。DE レジスタは常に保存されます。ただしいずれの場合も、裏レジスタ（AF'、BC'、DE' および HL'）とインデックスレジスタ（IX および IY）は破壊されます。

マッパーサポートの拡張 BIOS で利用できるファンクションは以下のとおりです。

- マッパー変数テーブルの獲得

パラメータ	A =0
	D =4（マッパーサポートのデバイス番号）
	E =1
結果	A =プライマリマッパーのスロットアドレス
	DE=保存される
	HL=マッパー変数テーブルの先頭アドレス

- マッパーサポートルーチンアドレスの獲得

パラメータ	A =0
	D =4
	E =2
結果	A =プライマリマッパーの総セグメント数
	B =プライマリマッパーのスロット番号
	C =プライマリマッパーの未使用セグメント数
	DE=保存される
	HL=ジャンプテーブルの先頭アドレス

これらのマッパーサポートの拡張 BIOS 自体では A=0 であることは特に必要ではありません。しかし、マッパーサポートルーチンが存在しない場合、レジスタ類は変更されずマッパーサポートルーチンが存在する場合、必ず A に 0 でない値が返されること

に注意して下さい。そのため、呼び出し時にA=0とし、返されたAレジスタの値を調べることによって、マッパーサポートルーチンの有無を判断することができます。

拡張BIOSによって返されるプライマリマッパーのスロットアドレスは、現在のページ3のRAMスロットアドレスと同じもので、通常の環境（Disk BASICおよびMSX-DOS）では ページ2にも同じRAMスロットが選択されています。MSX-DOSの場合、これはページ0および1についてもあてはまります。

15.3 マッパールーチンの使用法

プログラムは「ALL_SEG」ルーチンをコールすることによっていつでも16K単位でRAMセグメントを要求できます。このルーチンはプログラムが使用できる新しいセグメントのセグメント番号を返し、未使用セグメントが存在しない場合にはエラーを返します。プログラムで、基本の64KRAM以外のセグメントを使用する場合、必ず明示的に割り当てられたセグメントを利用して下さい。

セグメントの割り当てにはユーザーセグメントとシステムセグメントがあります。ユーザーセグメントはプログラムが終了した場合には自動的に解放されますが、システムセグメントはプログラムが明示的に解放しない限り解放されません。プログラムがセグメントを割り当てる場合には、必要のない限りユーザーセグメントとして割り当てて下さい。

RAMセグメントは指定のセグメントのバイト単位の読み書きをする「RD_SEG」および「WR_SEG」によってアクセスできます。「CALSEG」と「CALLS」は、既存のMSXシステムのインタースロットコールと同じような方法でセグメント間コールを実行します。

明示的にセグメントをページングしたり、特定のページ中にはどのセグメントがあるかを見つけるためのルーチンが提供されています。例えば、HL中のアドレスの最上位の2ビットによってページ（0~3）を指定するルーチン（「PUT_PH」と「GET_PH」）や、個別のページをアクセスするための専用ルーチン（「GET_Pn」および「PUT_Pn」）などです。これらのルーチンは非常に高速ですので、プログラムの性能が犠牲になることはありません。

ページ3はマッパーサポートルーチンとシステム変数を持っているため、セグメントの切り換えは絶対にできません。また、ページ0は割り込みおよびスロット切り換えのエントリポイントを持っているため、切り換えに際しては細心の注意が必要です。ページ1と2はどのようにも変更できます。

マッパーサポートルーチンはどれもスロット選択メカニズムにまったく影響を与えません。例えば、「PUT_P1」がコールされると、指定のRAMセグメントは、マッパースロットがページ1に選択されている場合にのみアドレス4000H~7FFFHに現れます。「RD_SEG」および「WR_SEG」ルーチンは指定のページ中の現在のスロットの選択に関わらず、常にRAMセグメントをアクセスできますが、ページ2はマッパーRAMスロットになっていなければなりません。

15.4 セグメントの割り付けと解放

[illegible] AF BC 以外のすべてのレジスタは保存されます。エラーはリターンの際にキャリーフラグがセットされることによって示されます。これらのルーチンがコールされるときにはスロットの選択と RAMのページングはどのような状態にあっても構わず、どちらも保存されます。スタックはこれらのルーチンのどちらがコールされる場合でも、ページ0あるいはページ2にあってはいけません。

プログラムはこれらのルーチンにより明示的にセグメントを割り付けない限り どのセグメントも（基本の64K を構成する4つのセグメントを除いて）使用してはならず、解放した後はそのセグメントを使用し続けてはなりません。

セグメントはユーザーセグメントあるいはシステムセグメントのどちらかとして割り当てることができます。両者の相違はプログラムが終了するとユーザーセグメントは自動的に解放されますが、システムセグメントは解放されないという点です。プログラム自体が終了した後でもセグメント中にデータを必要とする場合を除き、ユーザーセグメントとして割り付けて下さい。ユーザーセグメントは常に最小の番号の未使用セグメントから割り付けられシステムセグメントは最大の番号から割り付けられます。

「セグメントの割り付け」からのエラーは、通常未使用セグメントがないことを示しますが、A およびBレジスタに不正なパラメータが渡されたことを示す場合もあります。「セグメントの解放」からのエラーは指定のセグメント番号が存在しないか、あるいはすでに解放されていることを示します。

ALL_SEG (ALLocate SEGment)

パラメータ	A=0	ユーザーセグメントの割り付け
	A=1	システムセグメントの割り付け
	B=0	プライマリマッパーの割り付け
	B≠0	複数マッパーサポートによる割り付け
	FxxxSSPP	スロットアドレス（0の場合プライマリマッパー）
	xxx=000	指定のスロットのみ割り付け
	xxx=001	指定のスロット以外の割り付け
	xxx=010	指定のスロットで割り付けを試み、 失敗の場合他のスロット（あれば）を試みる
	xxx=011	指定のスロット以外で割り付けを試み 失敗の場合指定のスロットで試みる
結果	キャリーセット=未使用セグメントがない	
	キャリークリア=セグメントが割り付けられた	
	A=割り付けられたセグメント番号	
	B=マッパースロットのスロットアドレス (B=0 でコールされた場合は0)	

FRESEG (FREe SEGment)

パラメータ	A=解放するセグメント番号
	B=0 プライマリマッパー
	B≠0 プライマリ以外のマッパー
結果	キャリーセット=エラー
	キャリークリア=セグメントの解放に成功

15.5 インターセグメントリード・ライト

次の2つのルーチンは任意のマッパー RAM セグメントの1バイトのデータをリード・ライトするために用意されています。コール手順は MSX システム ROM によって提供されるインタースロットリード・ライトルーチンと非常に似ています。AF 以外のすべてのレジスタは保存され、セグメント番号が有効かどうかを確認するためのチェックはされません。

セグメント番号により任意の 16K のセグメントを4つのページのどれにも指定できるため、アドレスの上位2ビットは無視され、データは常にページ2を通してリード・ライトされます。データは現在のページングあるいはページ0および1におけるスロットの選択に関わらず、指定されたセグメントとのリード・ライトが行われますが、これらのルーチンがコールされるときには、マッパー RAM スロットがページ2に選択されていなければなりません。これは、アクセスを高速にするため、スロット切り換えを行わないからです。また、スタックはページ2にあってはなりません。これらのルーチンは割り込みを禁止して戻ります。

RD_SEG (ReaD SEGment)

パラメータ	A =読み出すセグメント番号
	HL=セグメント内のアドレス
結果	A =そのアドレスのバイトの値
	その他のすべてのレジスタは保存される

WR_SEG (WRite SEGment)

パラメータ	A =書き込むセグメント番号
	HL=セグメント内のアドレス
	E =書き込む値
結果	A =破壊される
	その他のすべてのレジスタは保存される

15.6 インターセグメントコール

インターセグメントコールをサポートするため、2つのルーチンが用意されています。これらは MSX のシステム ROM で提供されている2つのインタースロットコールをモデルと

しており その使用法も非常に似ています。

指定されたセグメントが実際に存在するかどうかについてのチェックは行われないため、これを確認するのはユーザーの責任となります。コールされたセグメントは指定されたアドレスのページにページングされますが、これらのルーチンはいずれもスロットの選択を変更しないため、マッパースロットがこのページでイネーブルされていることを確認するのはユーザーの責任になります。これによって、ルーチンは高速になります。

ページ3へのインターセグメントコールは実行することができません。これを実行しようとしても、ページングは行われず、指定されたアドレスが単にコールされるだけです。また、ページ0をコールする場合は、ページ0には割り込みやその他のエントリポイントがあるため、慎重に行なって下さい。これらのコールで スタックが切り換えられるページと重ならないように注意して下さい。

これらのルーチンはインタースロットコールと違ってコールされたルーチンへ制御を渡す前に割り込みを禁止するということはありません。したがってコールされたルーチンで割り込みフラグを変更しない限り、呼び出し側へは同じ状態で戻ります。

インターセグメントコールルーチンで内部的に使用されるレジスタIX、IY、AF'、BC' DE' HL' にはパラメータを渡すことはできません。これらのレジスタはインターセグメントコール またはコールされたルーチンで破壊されます。他のレジスタ（AF、BC、DEおよびHL）はコールされたルーチンへそのまま渡され、そこから呼び出し側へ返されます。

CAL_SEG (CALl SEGment)

パラメータ	IY=コールされるセグメント番号 IX=コールするアドレス AF BC、DE、HLがコールされたルーチンに渡される その他のレジスタは破壊される
結果	AF BC DE HL IXおよびIYがコールされたルーチンから返される。この他のすべては破壊される。

CALLS (CALL Segment)

パラメータ	AF BC DE HLがコールされたルーチンに渡される その他のレジスタは破壊される
コール手順	CALL CALLS DB SEGMENT DW ADDRESS
結果	AF、BC、DE、HL、IXおよびIYがコールされたルーチンから返される。この他のすべては破壊される。

15.7 ダイレクトページングルーチン

以下のルーチンはプログラムがハードウェアをアクセスする必要なしに、直接現在のページング状態を操作するために用意されています。このルーチンを使えば、ハードウェアの細かい部分に違いがあっても互換性が確保できます。これらのルーチンは非常に高速なため、これを使用することでプログラムの効率に影響することはありません。

ルーチンは任意の4つのマッパーレジスタに対する直接リード・ライトと同等の機能を提供するために用意されます。セグメント番号の有効性についてのチェックはされないため、ユーザーが行わなければなりません。ここで重要なことは、レジスタに書き込まれた値は、同時にメモリに記憶され、レジスタの値を要求されたときには、この記憶されているメモリの値を返し、決してメモリマッパーのレジスタを直接読み出すことはしないということです。

したがって、例えば「PUT_P1」によってセグメントをイネーブルし、「GET_P1」をコールすると実際に書き込んだ値が返されます。マッパーレジスタを直接読み込んだ場合には、セグメント番号のうち必要でない上位ビットは通常記録されないため、書き込んだ値とは異なる値が返される可能性があります。

また、システムに複数のマッパーレジスタが存在しているときに、ハードウェアの競合により誤動作する可能性があります。ですから、ユーザーは必ずこれらのルーチンを使用してメモリマッパーを操作しなければなりません。

「PUT_P3」ルーチンは提供されていますが、実際にはダミールーチンで、ページ3のレジスタは変更されません。これは、ページ3のレジスタの内容を絶対に変更してはならないためです。ただし、「GET_P3」ルーチンでページ3にどのセグメントがあるかを判断することはできます。

もう1組のルーチン(「GET_PH」と「PUT_PH」)は機能的には同じですが、ページをHレジスタの上位2ビットで指定します。これは、HLレジスタがアドレスを持っているときに有用です。これらのルーチンはHLレジスタを破壊しません。「PUT PH」はページ3のレジスタを変更しません。

PUT_Pn　n=0、1、2、3 ページの選択
　パラメータ　A=セグメント番号
　結果　なし
　　すべてのレジスタが保存される

GET_Pn　n=0、1、2、3 ページの選択
　パラメータ　なし
　結果　A=セグメント番号
　　すべての他のレジスタは保存される

PUT_PH
　パラメータ　H=アドレスの上位バイト
　　A=セグメント番号
　結果　なし
　　すべてのレジスタが保存される

GET_PH
　パラメータ　H=アドレスの上位バイト
　結果　A=セグメント番号
　　すべての他のレジスタは保存される

これらのダイレクトページングルーチンを使用してページングの状態を変更する前に、プログラムでは、まず「GET_Pn」を使用して、セグメントの初期状態を保存しておいて下さい（あとでそのセグメントを元に戻すため)。セグメントの初期状態はシステムの将来のバージョンでは変更される可能性があるので、プログラムでは、これらに固定した値を仮定してはなりません。

16章
エラー

新しいMSX-DOS2ファンクション（40H以上のファンクションコール）は「エラーコード」をAレジスタに返します。処理が成功するとゼロになります。ゼロでない場合、エラーコードはエラーの種類を示します。

MSX DOS2はファンクションコールからリターンする直前に「OR A」命令を実行するため、エラーが起こったかどうかテストするためには、外部プログラムの「CALL 5」命令の直後に「JRNZ」命令を使用します。このエラージャンプの飛び先では通常Bレジスタにエラーコードをロードして、「エラーコードを返して終了」ファンクション（P.320参照）を実行します。これはエラーコードをコマンドインタープリタに返し、そこで適当なメッセージが表示されます。

外部プログラムはMSX-DOS2のファンクションコールで返されたすべてのエラーについて、「エラーコードの説明」ファンクション（P.323参照）を利用して自分自身で実際のメッセージを得ることもできます。詳細については17.3「ファンクションの説明」を参照して下さい。

エラーコードは0FFHで始まり、値が下がってきます。40H未満の値は「ユーザエラー」で、システムでは使用されず、外部プログラムが独自のエラーを返すために使用することができます。コマンドインタープリタに返された20H未満のユーザエラーはメッセージを出力しません。

「エラーコードの説明」ファンクションコールがメッセージを持たないエラーコードを説明するように要求された場合、返される文字列は「Systemerror <n>」「システムエラー <n>」または「User error <n>」「ユーザエラー <n>」です。ここで<n>はエラー番号を示します。

以下に現在定義されているすべてのエラー番号とそのメッセージおよび意味を示します。またニーモニックも示されますが、これは通常ソースファイル中のシンボルとして使用され特定のエラーを参照するためにMSX-DOS2のシステム全体を通して使用しているものです。

16.1 ディスクエラー

このグループのエラーはディスクエラー処理ルーチンに渡されるものです。デフォルトでは、「中止、再試行」エラーとして報告されます。これらのエラーは「ディスクのフォーマット」以外の MSX DOS ファンクションコールでは、エラー処理ルーチンに渡されますので、BDOS からの戻り値としては返されません。

表 2.20 ディスクエラー一覧

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0FFH	**.NCOMP**	**Incompatible disk**	**このディスクは使用できません**
	そのドライブではディスクがアクセスできません（例えば片面ドライブで両面ディスクを読もうとしたとき）。		
0FEH	**.WRERR**	**Write error**	**書き込み異常です**
	ディスク書き込み中に起こるエラー。		
0FDH	**.DISK**	**Disk error**	**ディスクが異常です**
	原因不明のディスクエラー。		
0FCH	**.NRDY**	**Not ready**	**ディスクが入っていません**
	ディスクドライブが応答しません。通常ドライブ中にディスクがないことを表します。		
0FBH	**.VERFY**	**Verify error**	**正しく書き込まれませんでした**
	VERIFY が有効のとき、書き込みの後にセクタが正しく読めませんでした。		
0FAH	**.DATA**	**Data error**	**ディスクのデータが異常です**
	CRC エラーチェックが不正なためディスクセクタが読めませんでした。通常ディスクの損傷を表します。CRC は、ディスクの読み取りエラーを検出するために、データと一緒にディスクセクタに書き込まれます。この CRC によりディスクからデータが正しく読み取れたかを調べるのが、CRC エラーチェックです。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0F9H	**RNF**	**Sector not found**	**セクターが見つかりません**
	要求されたセクタがディスク上で見つかりませんでした。通常ディスクの損傷を表します。		
0F8H	**.WPROT**	**Write protected disk**	**ディスクが書き込み保護されています**
	書き込み禁止状態のディスクに書き込もうとしました。		
0F7H	**.UFORM**	**Unformatted disk**	**ディスクがフォーマットされていません**
	ディスクがフォーマットされていない、あるいは異なった記録方法を使用しています。		
0F6H	**.NDOS**	**Not a DOS disk**	**MSX-DOS ディスクではありません**
	ディスクが別のオペレーティングシステム用にフォーマットされており、MSX DO ではアクセスできません。		
0F5H	**WDISK**	**Wrongdisk**	**ディスクが違います**
	MSX-DOS がアクセスしている最中に別のディスクに交換されました。正しいディスクに交換しなければなりません。		
0F4H	**.WFILE**	**Wrong disk for file**	**このファイル用のディスクではありません**
	オープンされたファイルがディスク上にあるときに別のディスクに交換されました。正しいディスクに交換しなければなりません。		
0F3H	**.SEEK**	**Seek errror**	**シークエラーです**
	ディスクの要求されたトラックが見つかりませんでした。		
0F2H	**.IFAT**	**Bad file allocation table**	**FAT 異常です**
	ディスク上のファイルアロケーションテーブルが破壊されています。CHKDSK でディスク上のデータのいくつかを復活することができる場合があります。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0F1H	NOUPB		
	このエラーはディスク交換処理の一部としてMSX-DOS内部で常にトラップされユーザーに渡ることはないため、メッセージを持ちません。		
0F0H	IFORM	Cannot format this drive	**このドライブはフォーマットできません**
	フォーマットできないドライブをフォーマットしようとしました。通常RAMディスクをフォーマットしようとして起こります。		

16.2 MSX-DOS ファンクションエラー

以下のエラーは通常MSX DOSのファンクションコールで返されるものです。特定のMSX DOSのファンクションから返されるエラーの詳細については17.3「ファンクションの説明」を参照して下さい。

表 2.21 MSX-DOS ファンクションエラー一覧

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0DFH	.INTER	Internal error	**DOSが異常です**
	起こってはならないエラー。		
0DEH	.NORAM	Not enough memory	**メモリー不足です**
	MSX-DOSがその16Kのカーネルデータセグメントでメモリを使いつくしました。セクタバッファ数を減らすか、いくつかの環境変数を削除して下さい。RAMディスクを作成するための未使用セグメントがない場合にも起こります。		
0DCH	.IBDOS	Invalid MSX-DOS call	**無効なMSX-DOSファンクション番号です**
	MSX-DOSのコールが不正なファンクション番号で行われました。大部分の不正なファンクションコールはエラーを返しませんが、このエラーは「直前のエラーコードの獲得」ファンクションコール（P.249参照）が実行されると返される場合があります。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0DBH	**.IDRV**	**Invalid drive**	**無効なドライブ名です**

ドライブ番号のパラメータ、あるいはドライブ・パス・ファイル文字列中のドライブ文字が現在のシステムで存在しません。

0DAH .IFNM Invalid filename 不正なファイル名です

ファイル名文字列が不正です。これは、ドライブ・パス・ファイル文字列ではなく、純粋なファル名文字列についてのみ生成されます。

0D9H .IPATH Invalidpathname 無効なパス名です

ASCIIZ ドライブ・パス・ファイル文字列が渡されるすべてのファンクションコールによって返される可能性があります。文字列の構文がなんらかの形で不正であることを示します。

0D8H .PLONG Pathname too long パス名が長過ぎます

ASCIIZ ドライブ・パス・ファイル文字列が渡されるすべてのファンクションコールによって返される可能性があります。指定された完全なパス（使用されている場合にはカレントディレクトリも含む）が 63 文字よりも長いことを示します。

0D7H .NOFIL File not found ファイルが見つかりません

ディスク上のファイルを検索するすべてのファンクションによって、ファイルが見つからなかった場合に返されます。このエラーはディレクトリが指定され、それが見つからなかった時にも返されます。それ以外では次の.NODIR エラーが返されます。

0D6H .NODIR Directory not found ディレクトリが見つかりません

ドライブ・パス・ファイル文字列中の途中のディレクトリが見つからなかった場合に返されます。

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0DFH	.INTER	Internal error	DOSが異常です
0D5H	.DRFUL	Root directory full	ルートディレクトリがいっぱいです
	新しいエントリがルートディレクトリ中で要求され、それがすでにいっぱいである場合に、「作成」あるいは「移動」のファンクションによって返されます。ルートディレクトリは拡張できません。		
0D4H	.DKFUL	Disk full	ディスクがいっぱいです
	書き込まれようとしているデータの量に対してディスク上に十分な領域がない場合に、書き込み処理によって返されます。ディスクが一杯の場合にサブディレクトリを作成あるいは拡張しようとした場合にも起こります。		
0D3H	.DUPF	Duplicate filename	ファイル名が重複しています
	目的のファイル名がすでに目的のディレクトリ中にある場合、「名前の変更」あるいは「移動」ファンクションで起こります。		
0D2H	.DIRE	Invalid directorymove	ディレクトリが移動できません
	サブディレクトリをそれ自体の下に移動しようとしました。これはディレクトリ構造中で孤立したループを作成することになるため許されません。		
0D1H	.FILRO	Readonly file	ファイルが読み出し専用です
	「読み出し専用」属性ビットがセットされているファイルに書き込みあるいは削除しようとしました。		
0D0H	.DIRNE	Directory not empty	ディレクトリが空ではありません
	空でないサブディレクトリを削除しようとしました。		
0CFH	.IATTR	Invalid attributes	無効な属性です
	ファイルの属性を不正な方法で変更しようとしたか、あるいはサブディレクトリに対してのみ可能な処理をファイルに行おうとした場合に起こります。また、ボリューム名のFIB (File InformationBlock) の不正な使用によっても起こります。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0CEH	.DOT	Invalid . or .. operation	.や..に対しては操作できません
	サブディレクトリ中の「.」あるいは「..」エントリに対して、名前の変更や移動などといった不正な操作をしようとしました。		
0CDH	.SYSX	System fileexists	システムファイルが既にあります
	既存のシステムファイルとおなじ名前のファイルあるいはサブディレクトリを作成しようとしました。システムファイルは自動的には削除されません。		
0CCH	.DIRX	Directory exists	ディレクトリが既にあります
	既存のサブディレクトリとおなじ名前のファイルあるいはサブディレクトリを作成しようとしました。サブディレクトリは自動的には削除されません。		
0CBH	.FILEX	File exists	ファイルが既にあります
	既存のファイルとおなじ名前のサブディレクトリを作成しようとしました。ファイルはサブディレクトリを作成するときには自動的には削除されません。		
0CAH	.FOPEN	File is already in use	ファイルが使用中です
	そのファイルに対してすでにオープンされているファイルハンドルがあるファイルの削除、名前の変更、移動、あるいはその属性や日付や時刻の変更を、そのファイルハンドルを使用せずに行おうとしました。		
0C9H	.OV64K	Cannnot transfer above 64K	64K を越える転送はできません
	ディスク転送領域が0FFFFH を超えてしまいます。		
0C8H	.FILE	Fileallocation errror	ファイルの割当異常です
	ファイルのクラスタチェインが破壊されました。CHKDSK を使用して可能な限りファイルを復活させる必要があります。		
0C7H	.EOF	End of file	ファイルの終わりです
	ファイルポインタがすでにエンドオブファイルにある、あるいはそれを超えている場合に、さらにファイルから読み込もうとしました。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0C6H	**.ACCV**	**File access violation**	**ファイルアクセス異常です**
		適切なアクセスビットをセットしてオープンされたファイルハンドルに対して読み出し・書き込みを行おうとしました。標準ファイルハンドルのいくつかは読み出し専用あるいは書き込み専用モードでオープンされています。	
0C5H	**.IPROC**	**Invalid process id**	**無効なプロセスIDです**
		「親プロセスに戻る」ファンクション（P.318参照）に渡されたプロセスIDが不正です。	
0C4H	**.NHAND**	**No spare file handles**	**ファイルハンドルが足りません**
		すべてのファイルハンドルがすでに使用中である場合にファイルハンドルをオープンあるいは作成しようとしました。現バージョンでは64までファイルハンドルが利用できます。	
0C3H	**.IHAND**	**Invalid file handle**	**無効なファイルハンドルです**
		指定のファイルハンドルが、システムで許される最大のファイルハンドル番号よりも大きいです。	
0C2H	**.NOPEN**	**File handle not open**	**ファイルハンドルがオープンされていません**
		指定のファイルハンドルは現在オープンされていません。	
0C1H	**.IDEV**	**Invalid device operation**	**無効なデバイスオペレーションです**
		デバイスのファイルハンドルやFIBを、検索や移動などの不正な操作に使用しようとしました。	
0C0H	**.IENV**	**Invalid environment string**	**無効な環境変数です**
		環境変数名の文字列に不正な文字があります。	
0BFH	**.ELONG**	**Environment string too long**	**環境変数が長過ぎます**
		環境変数名あるいはその値の文字列が最大の255文字の長さを超えた、あるいは長すぎてユーザーバッファが足りません。	

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
0BEH	.IDATE	Invalid date	無効な日付けです

「日付のセット」に渡された日付のパラメータが不正です。

0BDH	.ITIME	Invalid time	無効な時間です

「時刻のセット」に渡された時刻のパラメータが不正です。

0BCH	.RAMDX	RAMdisk(driveH:) already exists	RAM DISK(ドライブ H:)は既にあります

RAM ディスクがすでに存在しているのに RAM ディスクを作成しようとした場合、「RAM ディスク」ファンクション（P.326参照）から返されます。

0BBH	.NRAMD	RAM disk does not exist	RAM DISK がありません

RAM ディスクが存在していないときに、ファンクションで RAM ディスクを削除しようとしました。存在しない RAM ディスクをアクセスしようとするファンクションでは、.IDRV エラーとなります。

0BAH	.HDEAD	File handle has been deleted	ファイルが消去されています

ファイルハンドルに関連したファイルが削除されたため、ファイルハンドルはもう使用できません。

0B9H	.EOL		

起こってはならない内部エラー。

0B8H	.ISBFN	Invalid sub-function number	無効なサブファンクション番号です

「デバイス I/O の制御」ファンクション（P.302参照）に渡されたサブファンクション番号が不正です。

0B7H	.IFCB	Invalid File Control Bolck	無効な FCB です

FCB を使ったファイルアクセスの際、.FOPEN などをコールせずに、無効な FCB を使用して読み書きした際に起きるエラーです。

16.3 プログラム終了エラー

以下のエラーはシステムによって内部的に生成され、「中断」ルーチンに渡されるエラーです。これは通常ファンクションコールからは返されません。中断ルーチンには外部プログラムが「エラーコードを返して終了」ファンクションコール（P.320参照）に渡す任意のエラーが渡されることに注意して下さい。

表2 22 プログラム終了エラー一覧

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
09FH	**STOP**	**Ctrl–STOP pressed**	**Ctrl–STOPが押されました**
	[CTRL]＋[STOP] キーは、すべての文字I/Oなどのほとんどのシステムコールでチェックされます。		
09EH	**.CTRLC**	**Ctrl–C pressed**	**Ctrl–Cが押されました**
	[CTRL]＋[C] は、ステータスチェックを実行する文字ファンクションの場合にのみチェックされます。		
09DH	**.ABORT**	**Disk operation aborted**	**ディスク入出力が打ち切られました**
	このエラーはディスクエラーが、ユーザーによって、あるいはシステムによって自動的に、中断されたときに起こります。元のディスクエラーコードは2次エラーコードとしてBレジスタから中断ルーチンに渡されます。		
09CH	**.OUTERR**	**Error on standard output**	**標準出力でエラーが起きました**
	標準出力チャンネルが文字ファンクション（ファンクション01H～0BH）を通してアクセスされている間に、エラーが起こった場合に返されます。元のエラーコードは2次エラーコードとしてBレジスタに入れられてアボートルーチンに渡されます。このエラーは通常、プログラムが標準ファイルハンドルを変更している場合にのみ起こります。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
09BH	.INERR	Error on standard input	標準入力でエラーが起きました
	標準入力チャンネルが文字ファンクション（ファンクション01H~0BH）を通してアクセスされている間に、エラーが起こった場合に返されます。元のエラーコードは2次エラーコードとしてBレジスタに入れられてアボートルーチンに渡されます。最も起こり得るエラーは「.EOF」エラーです。このエラーは通常、プログラムが標準ファイルハンドルを変更している場合にのみ起こります。		

16.4 コマンドエラー

以下のエラーはMSX-DOSのファンクションコールからは返されませんが、コマンドインタープリタによって使用されます。これらは外部プログラムから利用できますので、ここにのせてあります。9章「エラーおよびメッセージ」ではコマンドインタープリタでこれらのエラーが持つ意味の詳細が記述されています。

表 2.23 コマンドエラー一覧

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
08FH	.BADCOM	Wrong version of command	コマンドのバージョンが違います
	COMMAND2.COMがディスクからその非常駐部分をロードしたが、そのチェックサムが期待通りの値ではありません。		
08EH	.BADCM	Unrecognized command	コマンドが違います
	指定のコマンドが内部コマンドでなく、その名前の.COMおよび.BATファイルも見つかりませんでした。		
08DH	.BUFUL	Command too long	コマンドが長すぎます
	バッチファイル中のコマンドが127文字の長さを超えています。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
08CH	.OKCMD		
	COMMAND2.COM にコマンド行で渡したコマンドが終了した際に返される内部エラーコード（このエラーコードにはメッセージがありません）。		
08BH	.IPARM	Invalid parameter	**無効なパラメータです**
	コマンドへのパラメータが、なんらかの形で不正です。例えば範囲外の数値であるなど。		
08AH	.INP	Too manyparameters	**パラメータが多すぎます**
	コマンドが要求するすべてのパラメータを解析した後で、コマンド行上にまだ区切り文字でない文字が残っています。		
089H	.NOPAR	Missing parameter	**パラメータが不足しています**
	パラメータがあるべきところでエンドオブラインが見つかりました。		
088H	IOPT	Invalidoption	**無効なオプションです**
	コマンド行で/の後に指定された文字はそのコマンドでは不正です。		
087H	.BADNO	Invalid number	**無効な数値です**
	数値が求められているところに、数字以外の文字が入っています。		
086H	.NOHELP	File for HELP not found	**HELPファイルが見つかりません**
	ヘルプファイルが見つからなかった、あるいはパラメータが有効なHELPパラメータではありませんでした。		
085H	.BADVER	Wrongversion of MSX-DOS	**MSX-DOSのバージョンが違います**
	このエラーはコマンドインタープリタで使用されることはありません。コマンドインタープリタでこのエラーが発生した場合は自分自身で持っているメッセージを使用します。しかし、このエラーを返すことが有用な場合には、外部プログラムで使用することができます。		

エラーコード	ニーモニック	英語エラーメッセージ	日本語エラーメッセージ
084H	NOCAT	Cannot concatenatedestination file	複写先ファイルは結合できません
	CONCAT コマンドの目的のファイルがソースの指定に一致しています。		
083H	BADEST	Cannot create destination file	ファイルを作成できません
	COPY コマンドで目的のファイルが作成されると、ソースファイルのひとつ(あるいはすでに使用中の別のファイル)を重ね書きしてしまいます。		
082H	COPY	File cannot be copied onto itself	自分自身にはコピーできません
	COPY コマンドで目的のファイルが作成されるとソースファイルを重ね書きしてしまいます。		
081H	.OVDEST	Cannot overwrite previous destination file	ファイルの重ね書きができません
	COPY コマンドでワイルドカードを使ったソースが、ワイルドカードを使っておらず、ディレクトリでなく、デバイスでもない目的ファイルとともに指定されました。		

17章
ファンクションコール

この章ではMSX-DOS2のファンクションコールについて詳しく解説しています。ファイルハンドル、ファイル情報ブロック（FIB　File Information Block)、環境変数などといったシステムの特徴について概説している13章「ディスクファイルの構造」とともに読むことをおすすめします。

MSXではMSX-DOSとDisk-BASICという異なる環境でディスクを使用するので、MSX-DOSのファンクションコールを実行するには2つの方法があります。MSX-DOSの環境で実行される外部プログラムは、"CALL0005H"でファンクションを利用しなければなりません。Disk-BASICやDisk-BASICの環境で実行される他のMSXプログラム（通常ROMから実行される）では、"CALL 0F37DH"を通してファンクションを利用します。

F37Dhを通してシステムをコールするときには、特にエラーハンドリングやアボートルーチンを実行する場合に、いくつかの制限があります。また、(DISK-BASICの場合のように）マスターディスクROM中にない限り、ページ1にあるパラメータを渡すことはできません。なぜなら、このようなファンクションコールが実行されている間はマスターディスクROMがページ1に選択されているためです。

特定のファンクションについては、個々のファンクションの説明で相違点が明記してあります。

17.1　ファンクションコールの方法

MSX-DOSのファンクションコールはCレジスタにファンクションコードを、他のレジスタ（A、BC、DEおよびHL）に必要なパラメータを入れ、「CALL 5」命令を実行することによって行います。結果はそれぞれレジスタ中に返されます。

すべてのレジスタ（AF、BC、DEおよびHL）はMSX-DOSのコールによって破壊されるか、あるいは結果を返します。裏レジスタセット（AF'、BC'、DE'およびHL'）は常に保存され、インデックスレジスタ（IXとIY）は、これらが結果を返す場合を除いて保存されます。

MSX-DOSはコールされると内部スタックに切り換えるため、必要な外部プログラムのス

タックは8バイトだけです。

CP/Mとの互換性のため、CP/Mのファンクションに対応するすべてMSX-DOSのファンクションは、A=LおよびB=Hでリターンします。多くの場合、Aはゼロで成功を示し01Hあるいは0FFHで失敗を示すようなエラーフラグを返します。

新しいMSX-DOSのファンクション（40H以上のファンクションコードのもの）はエラーコードをAに、他の結果を別のレジスタ中に入れてリターンします。0のエラーコードはエラーなしを示し、コードがゼロでない場合は、なんらかのエラーが発生したことを示し、正確な原因はその値を調べることでつきとめることができます。エラーコードおよびメッセージのリストは9章で説明します。また、MSX-DOSでは「エラーコードの説明」ファンクション（P 323参照）が提供されていて、エラーコードについての説明文字列を与えます。

17.2　ファンクション一覧

以下にファンクションコールの全リストを示します。

「ファンクション」には、ファンクションの番号と機能を示します。

「CP/M互換」に「○」と記されているファンクションは、CP/M 2.2のファンクションと互換性があることを意味します。

「DOS1互換」に「○」と記されているファンクションは、MSX-DOS1と互換性があることを意味し、それ以外のファンクションは、MSX DOS2で新たに追加されたファンクションであることを意味します。

「エラー処理」に「○」のついたファンクションは、ユーザーのディスクエラー処理ルーチン（ファンクション64Hおよびファンクション70H参照）からコールできることを意味します。

「ページ」には、ファンクションの解説をしているページを示します。

表 2.24 MSX-DOS2 ファンクションコール一覧

	ファンクション	CP/M 互換	DOS1 互換	エラー 処理	ページ
00H	プログラムの終了	●	●		269
01H	コンソール入力	○	○	○	269
02H	コンソール出力	○	○	○	270
03H	補助入力	○	○	○	270
04H	補助出力	○	○	○	271
05H	プリンタ出力	○	○	○	271
06H	直接コンソールI/O	○	○	○	271
07H	直接コンソール入力		○	○	272
08H	エコーなしコンソール入力		○	○	272
09H	文字列出力	○	○	○	273
0AH	バッファ行入力	○	○	○	273
0BH	コンソールステータス	○	○	○	274
0CH	バージョン番号の獲得	○	○	○	274
0DH	ディスクリセット	○	○		275
0EH	ディスクの選択	○	○		275
0FH	ファイルのオープン [FCB]	○	○		275
10H	ファイルのクローズ [FCB]	○	○		277
11H	最初のエントリの検索 [FCB]	○	○		277
12H	次のエントリの検索 [FCB]	○	○		278
13H	ファイルの削除 [FCB]	○	○		279
14H	シーケンシャル読み出し [FCB]	○	○		279
15H	シーケンシャル書き込み [FCB]	○	○		280
16H	ファイルの作成 [FCB]	○	○		280
17H	ファイル名の変更 [FCB]	○	○		281
18H	ログインベクタの獲得	○	○	○	281
19H	カレントドライブの獲得	○	○	○	282
1AH	ディスク転送アドレスのセット	○	○		282
1BH	アロケーション情報の獲得		○		282

	ファンクション	CP/M 互換	DOS1 互換	エラー処理	ページ
21H	ランダム読み出し [FCB]	○	○		283
22H	ランダム書き込み [FCB]	○	○		283
23H	ファイルサイズの獲得 [FCB]	○	○		284
24H	ランダムレコードのセット [FCB]	○	○		284
26H	ランダムブロック書き込み [FCB]		○		285
27H	ランダムブロック読み出し [FCB]		○		286
28H	ゼロフィルを行うランダム書き込み [FCB]	○	○		286
2AH	日付の獲得		○	○	287
2BH	日付のセット		○	○	287
2CH	時刻の獲得		○	○	288
2DH	時刻のセット		○	○	288
2EH	ベリファイフラグのセット・リセット		○	○	289
2FH	アブソリュートなセクタの読み出し		○	○	289
30H	アブソリュートなセクタの書き込み		○	○	290
31H	ディスクパラメータの獲得			○	290
40H	最初のエントリの検索				291
41H	次のエントリの検索				292
42H	新しいエントリの検索				293
43H	ファイルハンドルのオープン				295
44H	ファイルハンドルの作成				296
45H	ファイルハンドルのクローズ				297
46H	ファイルハンドルの確保				297
47H	ファイルハンドルの複製				297
48H	ファイルハンドルからの読み出し				298
49H	ファイルハンドルへの書き込み				300
4AH	ファイルハンドルポインタの移動				301
4BH	デバイスのI/O制御				302
4CH	ファイルハンドルの検査				304
4DH	ファイルあるいはサブディレクトリの削除				304
4EH	ファイル名あるいはサブディレクトリ名の変更				305
4FH	ファイルあるいはサブディレクトリの移動				306

	ファンクション	CP/M 互換	DOS1 互換	エラー処理	ページ
50H	ファイル属性の獲得・セット				307
51H	ファイルの日付および時刻の獲得・セット				308
52H	ファイルハンドルの削除				309
53H	ファイルハンドルの名前の変更				309
54H	ファイルハンドルの移動				310
55H	ファイルハンドルの属性の獲得・セット				310
56H	ファイルハンドルの日付および時刻の獲得・セット				311
57H	ディスク転送アドレスの獲得			○	311
58H	ベリファイフラグ設定の獲得			○	312
59H	カレントディレクトリの獲得				312
5AH	カレントディレクトリの変更				313
5BH	パス名の解析				313
5CH	ファイル名の解析				314
5DH	文字の検査			○	315
5EH	パス文字列全体の獲得				316
5FH	ディスクバッファのフラッシュ				317
60H	子プロセスの起動				318
61H	親プロセスへ戻る				318
62H	エラーコードを返して終了				320
63H	アボート終了ルーチンの定義				320
64H	ディスクエラー処理ルーチンの定義				322
65H	直前のエラーコードの獲得			○	323
66H	エラーコードの説明			○	323
67H	ディスクのフォーマット				324
68H	RAM ディスクの作成あるいは破壊				326
69H	セクタバッファの割り付け				326
6AH	論理ドライブの割り当て			○	327
6BH	環境変数の獲得			○	328
6CH	環境変数のセット			○	328
6DH	環境変数の検索			○	329
6EH	ディスク検査ステータスの獲得・セット			○	329
6FH	MSX-DOS のバージョン番号の獲得			○	330
70H	リダイレクションのステータスの獲得セット			○	331

17.3 ファンクションの説明

以下にMSX-DOSのファンクションのそれぞれについて、古いもの、新しいものを含めて詳しく説明します。ファンクション番号の後のかっこ中の名前は「CODES.MAC」中で定義されているファンクションコードのパブリック・ラベルです。プログラムではこれらの名前を可能な限り使用しなければなりません。

40H未満のファンクションの多くはエラーコードではなくエラーフラグを返します。エラーフラグがセットされると、エラーの原因を示す実際のエラーコードは「直前のエラーコードの獲得」ファンクション（P.323参照）で得ることができます。40H以上のすべてのファンクションはAレジスタ中にエラーコードを返します。16章「エラー」では、ファンクションから返される一般的なエラーについて記述しています。本章での個々のファンクションの説明では、そのファンクション固有の主なエラーの状態について記述します。

ディスク上の情報を変更するファンクションコールの多くは自動的にディスクバッファをフラッシュしないため、ディスクはそのファンクションコール実行後すぐには、必ずしも正しく更新されないことに注意して下さい。このようなファンクションコールにはすべてのタイプの「作成」、「書き込み」、「削除」、「名前の変更」、「ファイル属性の変更」、「ファイルの日付と時刻の変更」ファンクションコールが含まれます。ディスクバッファを常にフラッシュするファンクションは、「バッファのフラッシュ」、「クローズ」、および「確保」だけです。これらの処理の後では、ディスクは常に正しく更新されています。

プログラムの終了

CP/M　DOS1

機能番号　00H(_TERM0)

コール手順　なし

戻り値　なし

解説　このファンクションは0のリターンコードでプログラムを終了させます。これは MSX-DOS1 および CP/M との互換性のために用意されており、プログラムを終了するには、「エラーコードを返して終了」ファンクションコール（P.320参照）を使用することを推奨します。この場合でも必要ならエラーがなかったことを示す0のエラーコードを渡すことができます。プログラムが終了するときに起こることについての詳細はそのファンクションコールの説明、および 12.2「MSX-DOSへのリターン」を参照して下さい。このファンクションコールは呼び出し側に戻ることはありません。

コンソール入力

CP/M　DOS1　エラー処理

機能番号　01H(_CONIN)

コール手順　なし

戻り値　L=A　キーボードからの文字

解説　文字が標準入力（ファイルハンドル0－通常はキーボード）から読み込まれ、標準出力（ファイルハンドル1　通常はスクリーン）へエコーされます。文字がなければ、文字が入力されるまで待ちます。「コンソールステータス」ファンクション（P.274参照）で指定されている様々なコントロール文字は種々の制御のためにトラップされます。この種の文字が検知されると、処理の後、別の文字を待ちます。したがって、このファンクションではそのコントロール文字はユーザーに返されることがありません。

コンソール出力

CP/M DOS1 エラー処理

機能番号 02H(_CONOUT)

コール手順 E 出力する文字

戻り値 なし

解説 Eレジスタに渡された文字が標準出力（ファイルハンドル1－通常はスクリーン）に書き出されます。プリンタエコーが有効になっていると、文字はプリンタにも出力されます。種々のコントロールコードやエスケープシーケンスは画面制御コードとして解釈されます。これらの一覧は14章「画面制御コード」に記載されており、標準のVT-52コントロールコードのサブセットになっています。TABはカラム位置が8の倍数になるまでスペースに展開されて出力されます。

コンソール入力ステータスのチェックが行われ、「コンソールステータス」ファンクション（P.274参照）で解説された特別なコントロール文字が現れた場合は、そこで解説したように処理されます。それ以外の文字はその後の「コンソール入力」ファンクションコールのために内部でセーブされます。

補助入力

CP/M DOS1 エラー処理

機能番号 03H(_AUXIN)

コール手順 なし

戻り値 L=A 入力文字

解説 文字が補助入力デバイス（ファイルハンドル3）から読み込まれ、文字がない場合には文字の入力を待ちます。補助入力デバイスはこのファンクションが使用される前にインストールされていなければなりません。デバイスがインストールされていないと、このファンクションは常にエンドオブファイル文字（^Z）を返します。

補助出力

CP/M　DOS1　エラー処理

機能番号	04H(_AUXOUT)
コール手順	E　出力する文字
戻り値	なし
解説	Eレジスタに渡された文字が補助出力デバイス（ファイルハンドル3）に書き出されます。補助出力デバイスは、このファンクションが使用される前にインストールされていなければなりません。デバイスがインストールされていないと、このファンクションは文字を捨ててしまいます。

プリンタ出力

CP/M　DOS1　エラー処理

機能番号	05H(_LSTOUT)
コール手順	E　出力する文字
戻り値	なし
解説	Eレジスタに渡された文字が標準プリンタデバイス（ファイルハンドル4－通常はプリンタ）へ送られます。コンソー[illegible]エコーする場合も、おなじチャンネルが使用されます。このファンクションではTABは展開されませんが、スクリーン出力が[CTRL]＋[P]プリンタにエコーされるときには展開されます。

直接コンソールI/O

CP/M　DOS1　エラー処理

機能番号	06H(_DIRIO)
コール手順	E　00H～FEH　出力する文字 　　FFH　　　入力要求
戻り値	A=L　入力 　00H　文字がない 　そうでない場合には入力文字。出力については未定義。

解　説　　E=FFHの場合には、標準入力（ファイルハンドル0）からの文字入力が調べられ、文字がないと00Hが返されます。文字があると、標準入力（ファイルハンドル0）から読み込まれ、エコーされず、またコントロール文字についてのチェックもされずにAレジスタに返されます。

E≠FFHの場合には、レジスタE中の文字がTABの展開やプリンタエコーされずに、標準出力（ファイルハンドル1）へ直接出力されます。また、このファンクションではコンソールステータスのチェックは行われません。またこのファンクション自体はTABを拡張しませんが、VT-52コントロールコードはTABの拡張を含むため、スクリーン上での効果は同じです。

直接コンソール入力

DOS1　エラー処理

機能番号　07H(_DIRIN)

コール手順　なし

戻り値　L=A　入力文字

解　説　　このファンクションは文字がない場合には文字の入力を待つことを除いて、ファンクション06Hの入力オプションと同一です。ファンクション06Hと同様、エコーやコントロール文字のチェックは行われません。このファンクションは、このファンクション番号を「I/Oバイトの取得」に使用しているCP/Mとの互換性はありません。

エコーなしコンソール入力

DOS1　エラー処理

機能番号　08H(_INNOE)

コール手順　なし

戻り値　L=A　入力文字

解　説　　このファンクションは入力された文字が標準出力にエコーされない点を除いて、「コンソール入力」ファンクション（P.269参照）と同様です。同様のコントロール文字のチェックが行われます。このファンクションは、このファンクション番号を「I/Oバイトの設定」に使用しているCP/Mとの互換性はありません。

文字列出力

CP/M DOS1 エラー処理

機能番号 09H(_STROUT)

コール手順 DE 文字列のアドレス

戻り値 なし

解説 DEレジスタで指される文字列の文字が通常の「コンソール出力」ファンクション（P.270参照）を使用して出力されます。文字列は「$」(ASCII 24H）で終了します。

バッファ行入力

CP/M DOS1 エラー処理

機能番号 0AH(_BUFIN)

コール手順 DE 入力バッファのアドレス

戻り値 なし

解説 DEは入力に使用されるべきバッファを指していなければなりません。このバッファの最初のバイトはバッファが保持できる文字数（0〜255）を持っていなければなりません。入力行は標準入力デバイス（ファイルハンドル0 通常キーボード）から読み込まれ、バッファに格納されます。入力はCRが標準入力から読み込まれたときに終了します。入力された文字数（CR自体は含まれない）は（DE+1）に格納されます。バッファに余裕があれば、CRが最後の文字の後に格納されます。

キーボードから入力する場合（通常の場合）、簡単な行エディタが利用でき、また、以前に入力された行の256バイトのリングバッファがあって、これを編集したり再入力したりすることができます。これらの編集機能についての詳細は2章「コマンド行の編集」を参照して下さい。入力バッファがいっぱいになると、バッファに入れられない文字がタイプされるたびに、コンソールのベルが鳴ります。入力された文字は標準出力に出力され、プリンタエコーが有効になっている場合は、プリンタにも出力されます。

コンソールステータス　CP/M　DOS1　エラー処理

機能番号　0BH(_CONST)

コール手順　なし

戻り値　L=A　00H　入力文字がない
　　　　　　FFH　入力文字がある

解説　キーボードからの入力について、入力文字があるかどうかを示すフラグがAレジスタに返されます。入力文字があればそれが読み込まれ、特殊なコントロール文字かどうか検査されます。そのようなコントロール文字でない場合には、内部の1バイトバッファに格納され、このファンクションへの以降のコールでは、キーボードをチェックしないで即座に「入力文字がある」を返します。このファンクションで「入力文字がある」と示された場合、その文字はいずれかの「コンソール入力」で読むことができます。

文字が「^C」であると、プログラムはユーザーのアボートルーチン（定義されている場合）を経由して「.CTRLC」エラーを返して終了します。文字が「^P」であると、プリンタエコーが有効になり、「^N」の場合に無効になります。文字が「^S」であると、ルーチンは別の文字が押されるのを待ち、それから「入力文字がない」という状態を返すことによって「ウェイト」機能を実現します。処理を続行するためにタイプされた文字は無視されますが、「^C」の場合にはプログラムが停止します。これらと同様の入力のチェックは、01H（P.269参照）、02H（P.270参照）、08H（P.272参照）、09H（P.273参照）、および0AH（P.273参照）でも実行されます。

バージョンの獲得　CP/M　DOS1　エラー処理

機能番号　0CH(_CPMVER)

コール手順　なし

戻り値　L=A　22H
　　　　H=B　00H

解説　このファンクションはエミュレートしているCP/Mのバージョン番号を返します。これは現在のシステムでは常にバージョン2.2です。

ディスクリセット

CP/M DOS1

機能番号 0DH(_DSKRST)

コール手順 なし

戻り値 なし

解説 内部バッファ中の、まだ書き出しが行われていないすべてのデータをディスクに書き出します。CP/M での場合のように、ディスクの交換を可能にするためにこのファンクションをコールする必要はありません。ディスク転送アドレスはまたこのファンクションによってその初期値 80H に戻されます。また、デフォルトドライブは A:となります。

ディスクの選択

CP/M DOS1

機能番号 0EH(_SELDSK)

コール手順 E ドライブ番号 (0=A: 1=B: など)

戻り値 L=A ドライブ数 (1~8)

解説 このファンクションは指定のドライブをデフォルトのドライブとして選択します。カレントドライブは、CP/M との互換性のため、0004H 番地にも格納されます。使用できるドライブの数を A レジスタに返します。ただし、ドライブ数に RAM ディスクは含みません。

ファイルのオープン [FCB]

CP/M DOS1

機能番号 0FH(_FOPEN)

コール手順 DE オープンされていない FCB へのポインタ

戻り値 L=A 0FFH ファイルが見つからない場合
0 ファイルが見つかった場合

解説 オープンされていない FCB にはドライブ (カレントドライブを示す場合には0) と ファイル名と拡張子 (ワイルドカードを使用してよい) が入っ

ていなければなりません。指定したドライブのカレントディレクトリで適合するファイルが検索され、見つかった場合にはそれがオープンされます。サブディレクトリやシステムファイルのエントリは無視され、ファイル名にワイルドカードを使用した場合には最初の適合するエントリがオープンされます。

デバイス名は（コロンなしで）FCB中に置くことができ、その場合にはデバイスをあたかもディスクファイルであるかのようにアクセスすることができます。標準のデバイス名は13.1「デバイスおよび文字I/O」で定義されています。

エクステント番号（extent number）の下位バイトはこのファンクションでは変更されず、ファイルはそれが指定された大きさを十分保持している場合にのみオープンされます。通常、アプリケーションプログラムではこのファンクションをコールする前にエクステント番号を0にセットします。エクステント番号の上位バイトは0にセットされ、CP/Mとの互換性を確保しています。

FCB中のファイル名と拡張子はディレクトリエントリからオープンされたファイルの実際の名前に置き換えられます。これは通常は元のものと同一ですが、ワイルドカードを使ったファイル名やファイル名中に小文字を使用している場合には異なることもあります。

レコードカウントは、1レコードあたり128バイトとしてファイルのサイズから計算した値を、指定のエクステント中のレコードサイズにセットします。ファイルサイズのフィールド、ボリュームID、および8つの予約バイトもセットアップされます。カレントレコードおよびランダムレコードのフィールドはこのファンクションでは変更されません。読み出しあるいは書き込みのファンクションを使用する前にそれらを初期化するのはアプリケーションプログラムの責任となります。

ファイルが見つからない場合には、環境変数「APPEND」が調べられます。これがセットされていると、ファイルを検索するべき2番目のディレクトリを指定するドライブ・パス文字列として解釈されます。指定のディレクトリでファイルを検索し、見つかった場合には前述のようにオープンします。この場合、元のFCBのドライブバイトがデフォルト（0）であれば、正しくファイルがアクセスできるようにドライブバイトをファイルが見つかったドライブにセットします。

ファイルのクローズ [FCB]

CP/M　DOS1

機能番号	10H(_FCLOSE)
コール手順	DE　オープンされたFCBへのポインタ
戻り値	L=A　0FFH　失敗した場合 0　成功した場合

解説

FCBはOPENまたはCREATファンクションコールのどちらかを使ってあらかじめオープンされていなければなりません。ファイルが読み出されただけであった場合、このファンクションは何もしません。ファイルに書き込みが行われていた場合には、バッファされていたすべてのデータがディスクに書き出され、ディレクトリのエントリが適切に更新されます。

ファイルはクローズ後もアクセスできるため、このファンクションは「確保」ファンクションと同等なものと考えることができます。

最初のエントリの検索 [FCB]

CP/M　DOS1

機能番号	11H(_SFIRST)
コール手順	DE　オープンされていないFCBへのポインタ
戻り値	L=A　0FFH　ファイルが見つからない場合 0　ファイルが見つかった場合

解説

このファンクションはFCB中の指定のドライブ（FCBのドライブ番号が0の場合はカレントドライブ）のカレントディレクトリで、FCB中のファイル名と拡張子に適合するファイルを検索します。ファイル名はワイルドカードが使用でき（「?」文字を含んでいる）、その場合には最初に適合するものが検索されます。エクステントフィールドの下位バイトが使用され、このエクステント番号を含むのに十分な大きさがあるファイルのみ見つかります。通常、このファンクションをコールする前に、エクステントフィールドをプログラムによって0にセットします。システムファイルおよびサブディレクトリのエントリは検索されません。

適合するものが見つかった場合（A=0）、ディレクトリエントリはDTA（ディスク転送アドレス）にコピーされ、前にドライブ番号が付けられます。これは、OPENファンクションコールでFCBとして直接使用するこ

とができます。エクステント番号は検索するFCBの下位バイトの値にセットされ、レコードカウントは適切に初期化されます(OPENの場合と同様)。ディレクトリエントリの属性バイトはS1バイトの位置にストアされますが、それはその通常の位置(ファイル名拡張子フィールドの直後)がエクステントバイトのために使用されるからです。

適合するものが見つからなかった場合(A=0FFH)、DTAは変更されません。いかなる場合も、DEで指される FCBはまったく変更されません。このファンクションはシステム内部で必要な情報を記憶しており、「次のエントリの検索」ファンクション(P.278参照)で検索を続行することができるため、「次のエントリの検索」ファンクションを実行する場合にFCBを保存しておく必要はありません。

CP/Mではこのファンクションでドライブ番号が「?」にセットされていると、すべてのディレクトリエントリ(割り当てられているものも解放されているものも)が適合します。また、エクステントフィールドが「?」にセットされるとファイルのすべてのエクステントが適合します。このような機能のどちらも、通常CP/Mのファイルシステム特有の特殊な目的のCP/Mプログラム(例えば「STAT」)でのみ使用されます。どちらの機能も、MSX DOS1およびMSX DOS2には存在しません。

次のエントリの検索 [FCB]

CP/M DOS1

機能番号　12H(_SNEXT)

コール手順　なし

戻り値
```
L=A  0FFH  ファイルが見つからない場合
     0     ファイルが見つかった場合
```

解説　このファンクションはファイル名に適合する次のファイルの検索を続行します。返される結果は「最初のエントリの検索」と同一ですので、そちらの説明を参照して下さい。検索を続行するために使用される情報はMSX-DOSの内部で保持され、「最初のエントリの検索」で使用した元のFCBはなくてかまいません。このファンクションは「最初のエントリの検索」ファンクションを実行した後でのみ使用可能です。

ファイルの削除 [FCB]

CP/M　DOS1

機能番号　13H(_FDEL)

コール手順　DE　オープンされていないFCBへのポインタ

戻り値　L=A　0FFH　ファイルがひとつも削除されなかった場合
　　　　　　0　　ファイルの削除が成功した場合

解説　FCBで指定したドライブのカレントディレクトリ中でFCB中のワイルドカードを使用したファイル名に適合するすべてのファイルが削除されます。サブディレクトリ、システムファイル、不可視、読み出し専用ファイルは削除されません。このファンクションは何らかのファイルの削除に成功した場合には、A=0として終了します。A=FFHでの終了はファイルがひとつも削除されなかったことを示します。

シーケンシャルな読み出し [FCB]

CP/M　DOS1

機能番号　14H(_RDSEQ)

コール手順　DE　オープンされたFCBへのポインタ

戻り値　L=A　01H　エンドオブファイルでエラーの場合
　　　　　　0　　読み出しが成功した場合

解説　このファンクションはファイルから次に続く128バイトのレコードを読み出し、現在のディスク転送アドレス（DTA）に入れます。レコードは現在のエクステント（上位および下位バイト）とカレントレコードによって指定されます。レコードの読み出しに成功すると、このファンクションはカレントレコードを1つふやし、それが80Hに達したらそれを0に戻して、エクステント番号を増やします。レコードカウントフィールドも　必要ならば常に更新されます。

MSX-DOSではCP/Mとは異なり、ファイルのサイズが128バイトの倍数である必要はないので、一部だけうめられたレコードができることがあります。この場合、断片レコードはそれが外部プログラムのDTAアドレスにコピーされるときに残りが0で詰められます。

シーケンシャルな書き込み [FCB] CP/M DOS1

機能番号 15H(_WRSEQ)

コール手順 DE オープンされたFCBへのポインタ

戻り値 L=A 01H ディスクが一杯でエラーの場合
0 書き込みが成功した場合

解説 このファンクションは現在のディスク転送アドレスから128バイトをファイルに書き出します。レコードはカレントレコードおよびエクステントによって指定されます。その後レコードおよびエクステントを適切に増やします。レコードカウントバイトはファイルが拡張された場合、または書き込みが新しいエクステントに移動した場合、正しく更新されます。FCB中のファイルサイズもファイルが拡張されたときには更新されます。

ファイルの作成 [FCB] CP/M DOS1

機能番号 16H(_FMAKE)

コール手順 DE オープンされていないFCBへのポインタ

戻り値 L=A 0FFH 失敗
0 成功

解説 このファンクションは指定のドライブのカレントディレクトリ中に新しいファイルを作成し、読み出しおよび書き込みのためにオープンします。ドライブ、ファイル名、およびエクステント番号の下位バイトはFCB中にセットアップされていなければなりません。その際に、ファイル名にワイルドカードを使用することはできません。また、不正なファイル名が生成されないようにチェックが行われます。

要求された名前のファイルがすでに存在する場合には、動作はエクステント番号バイトの値によって異なります。通常これは0で、その場合には古いファイルが削除されて新しいファイルが作成されます。エクステント番号が0でない場合には、新しいファイルは作成されず、既存のファイルがオープンされます。それぞれのエクステントが明示的に作成されなければならなかったCP/Mの初期のバージョンとの互換性がこれによって保証されます。

どの場合も、OPEN ファンクションコールが実行されたのと全く同様に、処理されたファイルは所定のエクステント番号にしたがってオープンされます。

ファイル名の変更 [FCB]

CP/M DOS1

機能番号 17H(_FREN)

コール手順 DE オープンされていない FCB へのポインタ

戻り値 L=A 0FFH 失敗
0 成功

解説 オープンされていない FCB は通常のドライブとファイル名、および (DE+17) から始まる2番目のファイル名を持ちます。最初のファイル名に適合する指定のドライブのカレントディレクトリ中のそれぞれのファイルは2番目のファイル名中にある「?」文字については対応する文字をもとのままにして、2番目のファイル名に変更されます。また、ファイル名の重複や不正なファイル名が作成されるのを防ぐため、チェックが行われます。サブディレクトリやシステムファイル、不可視の名前を変更することはできません。

ログインベクタの獲得

CP/M DOS1 エラー処理

機能番号 18H(_LOGIN)

コール手順 なし

戻り値 HL ログインベクタ

解説 このファンクションは利用できるそれぞれのドライブについて HL に対応するビットをセットして返します (L のビット 0 がドライブ「A:」に対応する)。8つまでのドライブ (「A:」から「H:」まで) が現在システムでサポートされ、H レジスタはリターン時には通常 0 になります。

カレントドライブの獲得

CP/M　DOS1　エラー処理

機能番号	19H(_CURDRV)
コール手順	なし
戻り値	L=A　カレントドライブ(0=A:など)
解説	このファンクションはカレントドライブの番号を返します。

ディスク転送アドレスのセット

CP/M　DOS1

機能番号	1AH(_SETDTA)
コール手順	DE　要求するディスク転送アドレス
戻り値	なし

解説

このファンクションはDEに渡されたアドレスをディスク転送アドレスとしてセットします。このアドレスはすべてのその後のFCBの読み出しおよび書き込みコールで使用され、「最初のエントリの検索[FCB]」(P.277参照)、「次のエントリの検索[FCB]」コール(P.278参照)ではディレクトリエントリをストアするために使用され、また「アブソリュートなセクタの読み出し・書き込み」コール(P.289、290参照)で使用されます。MSX-DOS2で新たに追加された「読み出し」および「書き込み」ファンクションでは使用されません。アドレスは「ディスクリセット」コール(P.275参照)によって80Hに戻されます。

アロケーション情報の獲得

DOS1

機能番号	1BH(_ALLOC)
コール手順	E　ドライブ番号(0=カレント　1=A:など)
戻り値	A　1クラスタあたりのセクタ数

BC　セクタサイズ(常に512)
DE　ディスク上のクラスタの総数
HL　ディスク上の未使用クラスタ数
IX　DPBへのポインタ
IY　最初のFATセクタへのポインタ

解　説　　このファンクションは指定のドライブ中のディスクについての様々な情報を返します。このファンクション番号をアロケーションベクタのアドレスを返すために使用している CP/M との互換性はありません。MSX-DOS1と異なり、FAT の最初のセクタだけが IY 中のアドレスからアクセスでき、そこのデータは次の MSX-DOS コールまでしか有効ではありません。

ランダムな読み出し [FCB]　CP/M　DOS1

機能番号　21H(_RDRND)

コール手順　DE　オープンされた FCB へのポインタ

戻り値　L=A　01H　エンドオブファイルでエラーの場合
　　　　　　　0　　読み出しが成功の場合

解　説　　このファンクションはファイルから 128 バイトのレコードを読み出して現在のディスク転送アドレス（DTA）に入れます。ファイルの位置は FCB の 3 バイトのランダムレコード番号（21H~23H）で決まります。CP/M と異なり、ランダムレコード番号の 3 バイトすべてが使用されます。ファイルの終わりにある断片レコードはユーザーの DTA にコピーされる前に 0 で詰められます。

ランダムレコード番号は変更されないため、このファンクションへの連続的なコールでは、外部プログラムがランダムレコード番号を変更しない限り同じレコードを読みます。副作用として、カレントレコードおよびエクステントはランダムレコード番号と同じレコードを参照するようにセットアップされます。つまり、連続的な読み出し（あるいは書き込み）をランダムな読み出しの後で実行でき、その場合には同じレコードから始まるということです。レコードカウントバイトもそのエクステントに対して正しくセットアップされます。

ランダムな書き込み [FCB]　CP/M　DOS1

機能番号　22H(_WRRND)

コール手順　DE　オープンされた FCB へのポインタ

戻り値　L=A　01H　ディスクが一杯でエラーの場合
　　　　　　　0　　エラーが起きなかった場合

解　説

このファンクションは現在のディスク転送アドレス（DTA）から128バイトのレコードをファイルに書き込みます。書き込みは3バイトのランダムレコード番号（21H～23H）で指定されるレコードの位置に行われます。ランダムレコード番号は3バイトすべてが使用されます。レコードの位置が現在のエンドオブファイルを超えている場合、初期化されていないディスク領域がその間を埋めるために割り当てられます。

ランダムレコード番号のフィールドは変更されませんが、カレントレコードおよびエクステントのフィールドは同じレコードを参照するようにセットアップされます。レコードカウントバイトは、ファイルが拡張された場合、あるいは書き込みが新しいエクステントにまで及んだ場合には、必要に応じて調整されます。

ファイルサイズの獲得 [FCB]　CP/M　DOS1

機能番号　23H(_FSIZE)

コール手順　DE　オープンされていないFCBへのポインタ

戻り値　L=A　0FFH　ファイルが見つからない場合
　　　　　　0　　ファイルが見つかった場合

解　説

このファンクションは「ファイルのオープン」（P.275参照）とまったく同じように、FCB中のファイル名に最初に一致するものを検索します。見つかったファイルのサイズは128バイト単位に切り上げられ、レコード数が決定されます。FCBの3バイトのランダムレコード領域がレコード数にセットされるため、それは存在しない最初のレコードの番号となります。ランダムレコード番号の4番目のバイトは変更されません。

ランダムレコードのセット [FCB]　CP/M　DOS1

機能番号　24H(_SETRND)

コール手順　DE　オープンされたFCBへのポインタ

戻り値　なし

解　説

このファンクションはFCB中の3バイトのランダムレコードフィールドをカレントレコードおよびエクステント番号によって決定されるレコードに設定します。ランダムレコード番号の4バイト目は変更されません。また、レコードがファイル中に実際に存在するかどうかについてのチェックは行いません。

ランダムなブロックの書き込み [FCB]　DOS1

機能番号　26H(_WRBLK)

コール手順　DE　オープンされたFCBへのポインタ
HL　書き込むレコード数

戻り値　A　01H　エラー
0　成功

解　説

現在のディスク転送アドレス（DTA）からランダムレコード番号によって決まるファイル中の位置へデータを書き込みます。ファイルをオープンした後でしかもこのファンクションをコールする前に、ユーザーによって設定されるFCB中のレコードサイズ領域（0EH [illegible]）でレコードのサイズは決まります。レコードのサイズが64バイトよりも小さい場合、ランダムレコード番号の4バイトすべてが使用され、それ以外の場合には最初の3バイトだけが使用されます。

書き込むレコード数はHLによって指定し、レコードサイズとこれとで書き込むデータの合計が決まります。これが64Kを越える場合にはエラーが返されるため、転送の最大サイズが制限されます。

データを書き込んだ後、ランダムレコード領域はファイル中の次のレコード番号に修正（つまり、HLがそれに加算）されます。カレントレコードおよびエクステントの領域は使用されず、変更もされません。ファイルが拡張された場合にはファイルサイズ領域が更新されます。

レコードサイズは1〜0FFFFHの任意の値です。小さいレコードサイズは大きいレコードサイズに効率では劣らないため、レコードサイズは1にセットすることもでき、その場合にはレコードカウントはバイトカウントになります。1回に大量の転送を行う方がこれを何回かの少量の転送で行うよりも速いため、1回のファンクションコールでできる限り多くのバイト数の書き込みを行う方が速度的に望ましいでしょう。

書き込むレコード数（HL）が0の場合、データは書き込まれず、ファイルのサイズがランダムレコード領域で指定した値に変更されます。これは

ファイルの現在のサイズより大きくても小さくてもかまわず、必要に応じてディスクの領域が割り当てられたり解放されたりします。この方法で割り当てられた追加のディスク領域は、特定の値に初期化されることはありません。

ランダムなブロックの読み出し [FCB]

DOS1

機能番号 27H(_RDBLK)

コール手順 DE オープンされたFCBへのポインタ
HL 読み出すレコード数

戻り値 A 01H エラー、通常エンドオブファイル
0 成功
HL 実際に読んだレコード数

解説 このファンクションは上記のブロックの書き込みファンクションと対をなすものであり、その使用法もほとんど同じです。この場合も大きなブロックを読むと、通常のCP/Mの処理よりもかなり速くなります。

例えば、ファイルから20Kを読みたい場合、1Kごとに20回の別々のファンクションコールを行うより、1回のファンクションコールで20Kを読む方がよい方法と言えます。ただし20Kの読み込みをレコードサイズ1 レコードカウント20Kとして行うか、レコードサイズ20K、レコードカウント1として行うか、あるいはその中間の任意の組み合せで行うかということに関しては違いはありません。

実際に読んだレコードの数がHLに返されます。エンドオブファイルになった場合、(この場合、断片レコードになれば、ユーザーのDTAにコピーする前に0で詰められます) この数は要求されたレコードの数よりも小さいことがあります。ランダムレコード領域は読まれていない最初のレコードに修正 (つまり、HLに返される値がそれに加算) されます。

ゼロフィルを行うランダムな書き込み [FCB]

CP/M DOS1

機能番号 28H(_WRZER)

コール手順　DE　オープンされたFCBへのポインタ

戻り値　L=A　01H　エラー
　　　　　　0　成功

解説　このファンクションは「ランダムな書き込み」(P.283参照) と同じですが、ファイルを拡張しなければならない場合に、データが書き込まれる前に新たに割り当てられたすべてのディスククラスタが0で埋められます。

日付の獲得

DOS1　エラー処理

機能番号　2AH(_GDATE)

コール手順　なし

戻り値　HL　年　1980~2079
　　　　D　月　1=1月~12=12月
　　　　E　日　1~31
　　　　A　曜日　0=日曜~6=土曜

解説　このファンクションは上記のような形式で内部のカレンダーの現在の値を返します。

日付のセット

DOS1　エラー処理

機能番号　2BH(_SDATE)

コール手順　HL　年　1980~2079
　　　　D　月　1=1月~12=12月
　　　　E　日　1~31

戻り値　A　00H　日付が有効
　　　　　FFH　日付が無効

解説　与えられた日付は有効性をチェックされ、有効ならば新しい日付としてセットされます。有効性のチェックはそれぞれの月の日数や閏年についても完全に行われます。日付が無効の場合、現在の日付は変更されません。日付はリアルタイムクロックチップに保持されるため、マシンの電源を切っても更新されています。

時刻の獲得

DOS1　エラー処理

機能番号　2CH(_GTIME)

コール手順　なし

戻り値
H　時間　0～23
L　分　0～59
D　秒　0～59
E　1/100秒　常に0

解説　このファンクションは上記の形式でシステムクロックの現在の値を返します。

クロックチップが1/100秒単位で計時できないため、Eレジスタには常に0が返されます。

時刻のセット

DOS1　エラー処理

機能番号　2DH(_STIME)

コール手順
H　時間　0～23
L　分　0～59
D　秒　0～59
E　1/100秒　無視される

戻り値
A　00H　時刻が有効
　FFH　時刻が無効

解説　このファンクションは内部のシステムクロックを指定の時刻の値にセットします。時刻が無効であると、Aレジスタは0FFHとして返されてエラーを示し、現在の時刻は変更されません。時刻はリアルタイムクロックチップに保持されるため、マシンの電源を切っても更新されます。

クロックチップが1/100秒単位で設定できないため、Eレジスタの値は無視されます。

ベリファイフラグのセット・リセット

DOS1 エラー処理

機能番号 2EH(VERIFY)

コール手順 E 0 ベリファイを無効に
0以外 ベリファイを有効に

戻り値 なし

解説 このファンクションはすべての書き込みの際の自動的なベリファイを有効あるいは無効にします。MSX DOSが起動したときにはデフォルトではオフになっています。ベリファイを有効にするとシステムの信頼性は向上しますが、同時に書き込みのスピードが低下します。また、この機能はディスクドライバに依存するため、ドライバがサポートしていないとベリファイ動作は行われません。

アブソリュートなセクタの読み出し

DOS1 エラー処理

機能番号 2FH(_RDABS)

コール手順 DE セクタ番号
L ドライブ番号 0=A:など
H 読み出すセクタ数

戻り値 A エラーコード 0=エラーなし

解説 このファンクションはセクタをファイルとして解釈することなしに、ディスクからセクタを直接読み出します。セクタ番号をディスク上の物理的な位置に変換するため、ディスクは有効なDOSのディスクでなければなりません。セクタは現在のディスク転送アドレスに読み出されます。ディスクエラーは通常の方法で知らされます。

アブソリュートなセクタの書き込み　DOS1　エラー処理

機能番号　30H(_WRABS)

コール手順
DE　セクタ番号
L　ドライブ番号　0=A:など
H　書き込むセクタ数

戻り値
A　エラーコード

解説

このファンクションはセクタをファイルとして解釈することなしに、ディスクへセクタを直接書き込みます。セクタ番号をディスク上の物理的な位置に変換するため、ディスクは有効なDOSのディスクでなければなりません。セクタは現在のディスク転送アドレスから書き込まれます。ディスクエラーは通常の方法で知らされます。

ディスクパラメータの獲得　エラー処理

機能番号　31H(_DPARM)

コール手順
DE　ディスクパラメータ（32バイトのバッファ）へのポインタ
L　[illegible]号
0=カレントドライブ、1=A:など

戻り値
A　エラーコード
DE　保存される

解説

このファンクションは、指定のドライブ中のディスクのフォーマットに関する一連のパラメータをユーザープログラム内に確保されたバッファへ返します。これはアブソリュートなセクタの読み出しおよび書き込みを行うプログラムが、アブソリュートなセクタ番号を解釈する場合に有用です。返されるパラメータには外部プログラムが使用するのに便利なパラメータを提供するために、いくぶん余分な情報が含まれています。返されるパラメータブロックのフォーマットを以下に示します。

オフセット	意味
DE+1, 2	セクタサイズ（現在は常に512）
DE+3	クラスタごとのセクタ数
	（2のn乗。ただしnは1以上の整数）
DE+4, 5	予約セクタ数（通常1）
DE+6	FATの数（通常2）
DE+7, 8	ルートディレクトリのエントリ数
DE+9, 10	論理セクタの総数
DE+11	メディアディスクリプタバイト
DE+12	FATごとのセクタ数
DE+13~14	ルートディレクトリの最初のセクタ番号
DE+15~16	最初のデータのセクタ番号
DE+17~18	最大クラスタ番号
DE+19	ダーティディスクフラグ
DE+20~23	ボリュームID（1＝ボリュームIDなし）
DE+24~31	システム予約（現在は常に0）

ダーティディスクフラグは、ディスク中にUNDELコマンドで復活できるファイルがあることを示すフラグです。したがって、ファイルあるいはディレクトリにクラスタが割り当てられるとこのフラグはリセットされます。

最初のエントリの検索

機能番号　40H(_FFIRST)

コール手順
DE　ドライブ・パス・ファイル ASCIIZ文字列またはFIBポインタ
HL　検索ファイル名 ASCIIZ文字列（DE=FIBポインタである場合のみ）
B　検索属性
IX　新しいFIBへのポインタ

戻り値
A　エラーコード
(IX)　一致するエントリが入る

解説　文字列の「ドライブ・パス」部分、あるいはFIB（ファイル情報ブロック）は検索するディレクトリを指定します。ファイルを指定するFIBが渡されると、「.IATTR」エラーが返されます。文字列の「ファイル」部分、あ

るいはHLが指す検索ファイル名ASCIIZ文字列は、どういったファイル名を一致させるかを決定します。一致したものがない場合には「.NOFIL」エラーが返され、あった場合にはIXによって指されるFIBが一致するエントリの情報で満たされます。

このファイル名にはワイルドカード文字（「?」および「*」）を含めてもよく、その場合には最初に一致するエントリが返されます。ファイル名がヌル（DEで指されるASCIIZ文字列がヌルであるか「¥」で終わっているか、あるいはHLで指される検索ファイル名文字列がヌルの場合）であると、このファンクションはあたかもファイル名が「*.*」であるかのように振るまい、したがってすべての名前が一致します。

Bレジスタ中の属性バイトは一致するエントリのタイプを指定します。これが0だと、不可視でもシステムファイルでもないファイルだけが見つけられます。Bレジスタにディレクトリ、不可視あるいはシステムビットがセットされていると、これらの属性を持つエントリが通常のファイルとともに一致するようになります。Bレジスタの読み出し専用およびアーカイブビットは無視されます。

Bレジスタのボリューム名ビットがセットされると検索は排他的に行われ、ボリュームラベルのエントリだけが検索されます。この場合にはまたFIBとファイル名あるいはドライブ・パス・ファイル文字列はドライブの指定を除いて無視されます。つまり、ボリューム名は指定のファイル名に一致するかどうかに関わらず、もし存在すれば、ルートディレクトリ中で見つけられます。

DEがFIBを指している場合、必要ならIXも同一のFIBを指すことができます。この場合、一致するものが見つかると、新しいFIBは古いものを重ね書きします。

次のエントリの検索

機能番号　41H(_FNEXT)

コール手順　IX　以前の最初のエントリを検索するファンクションから返されたFIBへのポインタ

戻り値　A　エラー
(IX)　次の一致するエントリが入る

解説　このファンクションは「最初のエントリの検索」ファンクションコールの後でのみ使用しなければなりません。これは「最初のエントリの検索」

ファンクションコールに与えた（ワイルドカードを使用したような）ファイル名に一致する次のエントリをディレクトリで探します。それ以上一致するエントリが存在しない場合には「.NOFIL」エラーが返され、存在した場合には、FIBが新しい一致するエントリの情報で満たされます。

新しいエントリの検索

機能番号　42H(_FNEW)

コール手順
DE　ドライブ・パス・ファイル ASCIIZ 文字列または FIB ポインタ
HL　検索ファイル名 ASCIIZ 文字列（DE=FIB ポインタの場合のみ）
B　b0～b6=要求する属性、b7=新規作成フラグ
IX　「テンプレートファイル名」を保持している新しい FIB へのポインタ

戻り値
A　エラー
(IX)　新しいエントリが入る

解説

このファンクションは前述の「最初のエントリの検索」ファンクション（P291参照）と大変よく似ています。HL および DE 中のパラメータはまったく同じ方法で使用され、ディレクトリエントリを指定します。ただし選択したディレクトリで指定された名前に一致するエントリを検索する代わりに、新しいエントリがこの名前で作成されます。IX で指される FIB にはあたかも「最初のエントリの検索」コールで見つかったかのように新しいエントリについての情報がセットされます。

ファイル名中にワイルドカード文字（「?」あるいは「*」）があると、それらは IX で指される新しい FIB のファイル名の位置にある「テンプレートファイル名」からの適切な文字に置き換わります。その結果がそれでもワイルドカード文字が残ったり、あるいは不正であったりすると「.IFNM」エラーが返されます。これは、自動的に名前を変更するコピー処理を行う場合に有用です。

「最初のエントリの検索」と同様、ファイル名がヌルであると、「*.*」であるのとまったく同じに扱われます。このファンクションでの場合それは「テンプレートファイル名」が新しく作成されるファイル名として使用されることを意味します。

ルートディレクトリに領域がないと「.DRFUL」エラーが返されます。また、サブディレクトリを拡張する必要があるにもかかわらず、ディスクがいっぱいである場合には「.DKFUL」エラーが返されます。

Bレジスタ中に渡される属性バイトは新しいエントリに付与すべき属性です。ボリューム名ビットがセットされていると、ルー[illegible]にボリューム名が作成されます。ディレクトリビットがセットされていると、作成されるエントリはサブディレクトリとなり、そうでない場合にはファイルとなります。ファイルには、システム、不可視、および読み出し専用ビットがセットでき、サブディレクトリには不可視ビットがセットできます。ファイルは常にアーカイブ属性ビットをセットして作成されます。

ファイルは現在の日付と時刻で長さ 0 で作成されます。サブディレクトリの場合には 1 つのクラスタが割り当てられ、「.」および「..」エントリが適切に初期化されます。

指定した名前のエントリがディレクトリ中にすでに存在した場合、動作は「新規作成フラグ」(B レジスタのビット 7) 、およびエントリのタイプによって異なります。「新規作成フラグ」がセットされていると、常に「.FILEX」エラーが返されます。したがってこのフラグをセットすると既存のファイルが削除されないことが保証されます。

すでにエントリが存在し、「新規作成フラグ」がセットされていないと既存のエントリのタイプが調べられて、それが新しいファイルを作成するために削除できるかどうかを判断します。エントリが読み出し専用ファイル (「.FILRO」エラー)、システムファイル (「.SYSX」エラー)、あるいはサブディレクトリ (「.DIRX」エラー) である場合、また、このファイルに対してすでにオープンしているファイルハンドル (「.FOPEN」エラー) が存在する場合、エラーが返されます。サブディレクトリを作成しようとしているときは、通常のファイルであっても削除されません (「.FILEX」エラー)。

これらのエラー (「.FILEX」、「.FILRO」、「.SYSX」、「.DIRX」、「.FOPEN」) が発生すると、既存のエントリの情報が FIB に書き込まれます。この FIB はあたかも「最初のエントリの検索」ファンクションで返されたかのように使用することができます。

ファイルハンドルのオープン

機能番号	43H(_OPEN)
コール手順	DE ドライブ・パス・ファイル ASCIIZ 文字列または FIB ポインタ A オープンモード b0 セット 書き込み禁止 b1 セット 読み出し禁止 b2 セット 継承 b3～b7 必ずクリア
戻り値	A エラー B 新しいファイルハンドル

解説

ドライブ・パス・ファイル文字列あるいは FIB は通常、サブディレクトリあるいはボリューム名でなく、ファイルを参照しなければなりません。これがボリューム名であると「.IATTR」エラーが返されます。これがサブディレクトリであると「.DIRX」エラーが返されます。

ファイルが指定されたとすると、それは読み出しおよび書き込み、あるいはその両方（A レジスタ中のオープンモードによる）ができるようにオープンされ、そのための新しいファイルハンドルが B レジスタに返されます。利用できる最小のファイルハンドル番号が使用され、ファイルハンドルの領域がない場合（「.NHAND」エラー）、メモリ不足の場合（「.NORAM」エラー）にはエラーとなります。

A レジスタの「読み出し禁止ビット」がセットされると、ファイルハンドルからの読み出しは拒否され、「書き込み禁止ビット」がセットされていると書き込みが拒否され、どちらの場合も「.ACCV」エラーとなります。ファイルが読み出し専用の場合も書き込みが拒否されます（「.FILRO」エラー）。A レジスタの「継承ビット」がセットされていると、「子プロセスの起動」ファンクションコール（P.318参照）によって生成された新しいプロセスにファイルハンドルが受け継がれます。

デバイスのファイルハンドルが組み込みデバイス（例えば「CON」や「NUL」）のひとつに一致するファイル名を与えられてオープンされると、それは最初は常に ASCII モードでオープンされます。「デバイスの I/O 制御」（P.302参照）を使用して、これをバイナリモードに変更することができますが、エンドオブファイルの状態が存在しないため、デバイスからバイナリモードで読む場合には慎重に行わなければなりません。

ファイルハンドルの作成

機能番号	44H(_CREATE)

コール手順

DE　ドライブ・パス・ファイル ASCIIZ 文字列
A　オープンモード
　b0セット　書き込み禁止
　b1セット　読み出し禁止
　b2セット　継承
　b3〜b7　必ずクリア
B　b0〜b6　要求する属性、b7=新規作成フラグ

戻り値

A　エラー
B　新しいファイルハンドル

解説

ファイルまたはサブディレクトリ（Bレジスタの属性の指定による）がドライブ・パス・ファイル文字列によって指定される名前およびそのディレクトリ中に作成されます。Bレジスタがボリューム名を指定していると「.IATTR」エラーが返されます。

ファイルまたはサブディレクトリが作成できないとエラーが返されます。この場合のエラーの状態は、「新しいエントリの検索」ファンクション（P.291参照）のものと同じで、主なエラーコードは「.FILEX」、「.DIRX」、「.SYSX」、「.FILRO」、「.FOPEN」、「.DRFUL」、あるいは「.DKFUL」です。「新しいエントリの検索」ファンクションと同様、「新規作成フラグ」（Bレジスタのビット7）がセットされていると、既存のファイルは削除されず、「.FILEX」エラーが常に返されます。

属性バイトがサブディレクトリを指定している場合には、不可視ビットをセットして不可視のサブディレクトリを作成することもできます。ファイルの場合は不可視、システム、あるいは読み出し専用ビットをセットしてそれぞれの属性を持ったファイルを作成することができます。不正な属性ビットは無視されます。ファイルは常にアーカイブビットをセットして作成されます。

ファイルは前述の「ファイルハンドルのオープン」ファンクション（P.295参照）の場合と同様、自動的にオープンされ、ファイルハンドルがBレジスタに返されます。「オープンモード」パラメータは、「ファイルハントルのオープン」ファンクションの場合と同様に解釈されます。サブディレクトリはオープンされない（これをオープンするのは無意味）ため、Bレジスタに有効なファイルハンドルとはなり得ない0FFHが返されます。

ファイルハンドルのクローズ

機能番号　45H(_CLOSE)

コール手順　B　ファイルハンドル

戻り値　A　エラー

解説　このファンクションは指定のファイルハンドルを解放して再利用できるようにします。ハンドルにファイルが書き込まれていると、そのディレクトリエントリが新しい日付と時刻で更新され、アーカイブ属性ビットがセットされ、そしてすべてのバッファ中のデータがディスクに書き出されます。このファイルハンドルを続けて使おうとするとエラーとなります。「ファイルハンドルの複製」(P.297参照)や「子プロセスの起動」ファンクション(P.318参照)で作成されたこのファイルハンドルのコピーが別に存在する場合には、これらのコピーは続けて使用することができます。

ファイルハンドルの確保

機能番号　46H(_ENSURE)

コール手順　B　ファイルハンドル

戻り値　A　エラー

解説　ファイルハンドルにファイルが書き込まれていると、そのディレクトリエントリが新しい日付と時刻で更新され、アーカイブ属性ビットがセットされ、そしてすべてのバッファ中のデータがディスクに書き出されます。ファイルハンドルは解放されないため　それを使用して続けてファイルをアクセスすることができ、現在のファイルポインタの設定は変りません。

ファイルハンドルの複製

機能番号　47H(_DUP)

コール手順　B　ファイルハンドル

戻り値　A　エラー
B　新しいファイルハンドル

解　説

このファンクションは指定のファイルハンドルのコピーを作成します。利用できる最小のファイルハンドル番号が常に使用され、利用できるものがないと「.NHAND」エラーが返されます。新しいファイルハンドルは元のものと同一のファイルを参照し、そのどちらを使用してもかまいません。一方のハンドルのファイルポインタが移動されると他方もまた移動されます。どちらかのハンドルがクローズしても、他方は続けて使用できます。

このファンクションで作成される複製ファイルハンドルは「別々にオープンされた」ものではないため「.FOPEN」エラーを生成するような別々のファイルハンドルとは見なされないことに注意して下さい。例えば、「複製」ファイルハンドルは名前を変更したり（P.309参照）、その属性を変更したり（P.310参照）できますが、その効果は両方のハンドルにおよびます。

特に、「複製」ファイルハンドルのひとつのコピーが削除される（P.310参照）と、ファイルは実際に削除されるので、他方のファイルハンドルはオープンしたままの状態ですが、安全に使用することができなくなることに注意して下さい。それが（クローズ、確保、あるいは削除以外で）使用されると「.HDEAD」エラーが返されます。

ファイルハンドルからの読み出し

機能番号　48H(_READ)

コール手順
B　ファイルハンドル
DE　バッファアドレス
HL　読み込むバイト数

戻り値
A　エラー
HL　実際に読み込んだバイト数

解　説

指定されたバイト数をファイルの現在のファイルポインタの位置からDEレジスタで指定されたバッファアドレスへ読み込み、ファイルポインタはそれに続くバイトに更新されます。ファイルハンドルが「読み出し禁止」アクセスビットをセットしてオープンされた場合には、「.ACCV」エラーが返されます。

読み込まれたバイト数は要求された数よりも少ない場合もあり、エラーがない場合にはレジスタHLに読まれた数が返されます。一般的には、要求よりも少ない数が読み込まれた場合、エラー状態としては扱ってはならず、「.EOF」エラーが返されるまでは、続きを読むためにさらに読み出しを行わなければなりません。「.EOF」エラーは部分的な読み出しでは返さ

れることはなく、0バイトを読む読み出しの場合にだけ返されます。このようなファイルの読み出しはデバイスファイルハンドルが正しく動作することを保証します。

ディスクファイルでは、読まれたバイト数はエンドオブファイルに達した場合にのみ要求した数よりも少なくなり、この場合、次の読み出し処理が0バイトを読んで、「EOF」エラーを返します。デバイスファイルハンドル（例えば0から4の標準ファイルハンドル）から読み出すと、その動作はそれぞれのデバイスによって異なり、また、その読み出しのモードがASCIIかバイナリであるかによっても異なります（後述のファンクションを参照)。最も一般的に使用されるデバイスである「CON」デバイスを例にとって説明しますが、他のデバイスも同様に動作します。

「CON」デバイスからバイナリモードで読み出すと、変換されたり、画面やプリンタにエコーされることなしに、キーボードから文字が読み出されます。要求された正確な文字数が常に読まれ、エンドオブファイルの状態は存在しません。エンドオブファイルの状態が存在しないため、バイナリモードでデバイスを読み出すときには慎重に行わなければなりません。

「CON」デバイスから ASCII モード（デフォルトモードで、標準入力チャンネルに通常適用される）での読み出しファンクションコールは、入力を1行読むだけです。入力行は、通常の行の編集機能をユーザーに利用できるようにして、キーボードから読み込まれ、入力された文字は画面にエコーされ、^P が有効な場合にはプリンタにもエコーされます。特殊な制御文字「^P」、「^N」、「^S」、および「^C」が調べられ、「コンソールステータス」(P.274参照）の場合とまったく同様に処理されます。

ユーザーが ↵ キーを入力すると、行は読み出しバッファにコピーされ、CR-LF シーケンスが最後に付加され、適切なバイト数が読み出しファンクションから返されます。次の読み出しは、別のバッファ行入力処理を開始します。読み出しで要求されたバイト数が行入力の長さよりも小さい場合には、要求と同じ文字数が返され、次の読み出しファンクションコールは行をすべて読んでしまうまで、その行の残りの部分を即座に返します。

ユーザーが「^Z」文字で始まる行を入力すると、これはエンドオブファイルを示すものとして解釈されます。この行は捨てられ、読み出しファンクションコールは0バイトを読み、「EOF」エラーを返します。これに続く読み出しは、通常に戻り、別の行入力を始めます。このように、エンドオブファイルの状態は永続的なものではありません。

ハンドルへの書き込み

機能番号	49H(_WRITE)
コール手順	B ファイルハンドル DE バッファアドレス HL 書き込むバイト数
戻り値	A エラー HL 実際に書き込んだバイト数

解説

このファンクションは「読み出し」ファンクション（P.298参照）と大変似ています。指定のバイト数がファイル中の現在のファイルポインタの位置に書き込まれ、ファイルポインタは書き込まれた最終バイトの直後を指すように更新されます。ファイルが「書き込み禁止」アクセスビットをセットしてオープンされた場合には、「.ACCV」エラーが返され、ファイルが読み出し専用であると「.FILRO」エラーが返されます。

書き込みが現在のエンドオブファイルを超えて行われると、ファイルは必要に応じて拡張されます。ファイルポインタがすでにエンドオブファイルを超えていると、ディスク領域が間を埋めるために割り当てられ、初期化されません。ディスク領域が十分にないと、「.DKFUL」エラーが返され、データの一部を入れる余地があったとしても、データはまったく書き込まれません。

書き込まれたバイト数はエラーが返された場合には0、書き込みが成功した場合には要求した数と等しいため、通常は無視できます。多くの小さなブロックよりも少数の大きなブロックでファイルを書き込んだ方が効率的であるため、プログラムでは常にできる限り大きなブロックで書き込むようにした方が良いでしょう。

このファンクションはファイルハンドルに「変更有り」ビットをセットし、それによって、ファイルハンドルが明示的にせよ暗黙にせよクローズされたり確保されたときに、ディレクトリエントリが新しい日付と時刻、および新しい割り当て情報で更新されます。また、アーカイブビットがセットされ、このファイルが最後にアーカイブされた後に変更されたということを示します。

デバイスファイルハンドルへの書き込みは、エンドオブファイル状態や行入力を考慮しなくてもよいため、読み出しのように複雑ではありません。「CON」デバイスへの書き込みの際に、ASCIIとバイナリモードとの間には相違があり、コンソールステータスのチェックはASCIIモードでだけ実

行されます。プリンタエコーが有効の場合も、ASCIIモードでだけ実行されます。

ファイルハンドルポインタの移動

機能番号　4AH(_SEEK)

コール手順
B　ファイルハンドル
A　方式コード
DE:HL　符号付きオフセット

戻り値
A　エラー
DE:HL　新しいファイルポインタ

解説

指定したファイルハンドルに結合したファイルポインタを方式コードとオフセットにしたがって変更し、新しいポインタ値をDE:HLに返します。方式コードは符号付きオフセットがどこからの相対であるかを以下のように指定します。

A=0　ファイルの先頭から相対
A=1　現在の位置から相対
A=2　エンドオブファイルから相対

オフセット0を指定すると移動方式コード1は単に現在のポインタ値を返し、移動方式コード2はファイルの大きさを返すということに注意して下さい。エンドオブファイルのチェックは行われないので、ファイルポインタをエンドオブファイルを超えてセットすることは可能です。「ファイルハンドルの複製」(P.297参照) あるいは「子プロセスの起動」(P.318参照) で作成された、このファイルハンドルのコピーがある場合には、それらのファイルポインタも変更されます。

ファイルポインタはランダムアクセスが可能なディスクファイルでのみ実際に意味を持ちます。デバイスファイルでも、ファイルポインタはすべての読み出しあるいは書き込みの際適切に更新され、またこのファンクションで調べたり変更したりすることができます。しかし変更は効果がなくそれを調べることが役にたつ場合はまずありません。

デバイスのI/O制御

機能番号　4BH(_IOCTL)

コール手順
B　ファイルハンドル
A　サブファンクションコード
　00H　ファイルハンドルの状態の獲得
　01H　ASCII・バイナリモードのセット
　02H　入力レディの検査
　03H　出力レディの検査
　04H　画面サイズの検出
DE　他のパラメータ

戻り値
A　エラー
DE　他の結果

解　説

このファンクションによって、ファイルハンドルの種々の状態を調べたり変更したりすることができます。特に、これを使用してファイルハンドルがディスクファイルを参照しているのかあるいはデバイスを参照しているのかを判断することができます。これは、ディスクファイルとデバイスI/Oで異なった動作をしたいプログラムで役に立ちます。

このファンクションはBレジスタにファイルハンドル、Aレジスタに処理を指定するサブファンクションコードを渡して実行します。特定のサブファンクションで必要となる他のすべてのパラメータはDEレジスタに渡し、結果はDEレジスタに返ります。サブファンクションコードが不正だと、「.ISBFN」エラーが返されます。

A=0だと処理は「ファイルハンドルステータスの獲得」となります。これは、ファイルハンドルについての種々の情報を与える1ワードのフラグを返します。この1ワードのフォーマットはデバイスファイルハンドルとディスクファイルハンドルでは異なり、ビット7がどちらかを指定します。このワードのフォーマットは次のとおりです。

● デバイスの場合

	ビット	意味
DE	0	セット=コンソール入力デバイス
	1	セット=コンソール出力デバイス
	2~4	システム予約
	5	セット=ASCII モード クリア=バイナリモード
	6	セット=エンドオブファイル
	7	常にセット（=デバイス）
	8~15	システム予約

● ディスクファイルの場合

	ビット	意味
DE	0~5	ドライブ番号（0=A:など）
	6	セット=エンドオブファイル
	7	常にクリア（=ディスクファイル）
	8~15	システム予約

エンドオブファイルフラグがデバイスの場合とディスクファイルの場合で同じ位置にあることに注意して下さい。デバイスの場合には、デバイスからの直前の読み出しが「EOF」エラーを生成したときにセットされ、次の読み出しによってクリアされます。ディスクファイルの場合には、ファイルポインタとファイルサイズを比較することによって作り出されます。

A=1のとき、処理は「ASCII・バイナリモードのセット」となります。この処理はデバイスファイルハンドルの場合にだけ実行できます。ASCII バイナリフラグはレジスタEのビット5（「ファイルハンドルステータスの獲得」によってそれが返されるところ）に渡しておかなければなりません。これはASCIIモードではセット、バイナリモードではクリアとなります。DEレジスタの他のすべてのビットは無視されます。

A=2または3のとき、処理はそれぞれ「入力レディの検査」または「出力レディの検査」となります。どちらの場合も、フラグかレジスタEに返され、ファイルハンドルがレディであるとFFH、そうでないと00Hとなります。「レディ」の意味はデバイスによって異なります。ディスクファイルハンドルは常に出力についてレディであり、ファイルポインタがエンドオブファイルにない限り入力についてレディです。「CON」デバイスはキーボードの状態をチェックして、それが入力についてレディであるかどうかを判断します。

A=4のとき、処理は「画面サイズの獲得」になります。これは行数をDレジスタに、列数をEレジスタに入れて、そのファイルハンドルの論理画面サイズを返します。画面サイズがないデバイス（ディスクファイルなど）では、DもEも0になります。したがって、どちらが0でも、それは「無限」と解釈します。例えば、このファンクションは1行に表示するファイルの数を判断するために「DIR /W」コマンドで使用され、Eレジスタの値が0の場合、デフォルトの80として扱います。

ファイルハンドルのテスト

機能番号　4CH(_HTEST)

コール手順
B　ファイルハンドル
DE　ドライブ・パス・ファイル ASCIIZ文字列またはFIBポインタ

戻り値
A　エラー
B　00H　同じファイルでない
　　FFH　同じファイル

解説
この特殊なファンクションには、ファイルハンドルとファイルを識別するドライブ・パス・ファイル文字列あるいはFIBブロックを渡します。このファンクションは2つのファイルが実際に同じファイルかどうかを判断し、その結果を示すフラグを返します。ファイルハンドルがディスクファイルでなくデバイスのものである場合には、「同一のファイルでない」ことを示す「B=00H」が常に返ります。

このファンクションによって、COPYコマンドがファイルをそのファイル自身の上にコピーしてしまうなどといったある種のエラーの状態を検知することができ、ユーザーにそうした情報をエラーメッセージとして表示することが可能です。これは、同様の検査が必要な他のプログラムでも利用できます。

ファイル・サブディレクトリの削除

機能番号　4DH(_DELETE)

コール手順
DE　ドライブ・パス・ファイル ASCIIZ文字列またはFIBポインタ

戻り値
A　エラー

解説

このファンクションはドライブ・パス・ファイル文字列あるいはFIBで指定されるもの（ファイルあるいはサブディレクトリ）を削除します。ワイルドカード文字は許されず、このファンクションではひとつだけのファイルまたはサブディレクトリが削除できます。サブディレクトリは空である場合にのみ削除でき、空でないとエラー（「.DIRNE」）となります。サブディレクトリ中の「.」と「..」のエントリは削除できず（「.DOT」エラー）、ルートディレクトリも削除できません。ファイルはオープンされているファイルハンドルがある場合（「.FOPEN」エラー）、あるいは読み出し専用である場合（「.FILRO」エラー）には、削除することができません。

ファイルの場合、それに対して割り当てられていたすべてのディスク領域が解放されます。ディスクがMSX DOS2のディスクであると、UNDELコマンドでファイルを復活するのに十分な情報がディスク上に残されます。この情報はそのディスクで次にディスク領域の割り当て（通常ファイルへの書き込み）が実行されるまで残されます。FIBが渡された場合には、このファンクションの実行後は、それを「次のエントリの検索」ファンクションに渡す場合以外には使用してはいけません。なぜならそれが参照しているファイルはもう存在しなくなっているからです。

「CON」などのデバイス名が指定されるとエラーは返されませんが、そのデバイスは実際には削除されません。

ファイル名・サブディレクトリ名の変更

機能番号　4EH(_RENAME)

コール手順　DE　ドライブ・パス・ファイル ASCIIZ文字列または FIB ポインタ
HL　新しいファイル名 ASCIIZ 文字列

戻り値　A　エラー

解説

このファンクションはドライブ・パス・ファイル文字列あるいはFIBで指定されるファイルあるいはサブディレクトリを、HLで指される文字列中の新しい名前に変更します。新しいファイル名文字列にドライブ文字やディレクトリパスを含めると「.IFNM」エラーになります。「CON」などのデバイス名が指定されるとエラーは返されませんが、そのデバイスは実際には名前の変更はされません。

ワイルドカード文字はドライブ・パス・ファイル文字列中では使用できません。したがってこのファンクションで名前を変更できるファイルあるいはサブディレクトリはひとつだけです。しかし、ワイルドカード文字はHL中

に渡される新しいファイル名では使用でき、それが該当する位置では、既存のファイル名文字が変更されずに残ります。その際には、不正なファイル名が作成されるのを避けるため、チェックが行われます。例えば、「XYZ」というファイルは新しいファイル名文字列「????A」へは変更できません（新しいファイル名は「XYZA」となり、不正であるため）。この場合には「.IFNM」エラーが返されます。

すでに新しいファイル名と同じエントリが存在する場合には、エラー（「.DUPF」）が返されて重複したファイル名を作成するのが回避されます。サブディレクトリ中の「.」と「..」のエントリは名前を変更できず（「.IDOT」エラー）、ルートディレクトリ名の変更もできません（ルートには名前がない）。オープンされているファイルハンドルを持つファイル（「.FOPEN」エラー）はファイル名を変更できませんが、読み出し専用ファイルは名前を変更できます。

DEがFIBを指していると、これは新しいファイル名に更新されないことに注意して下さい。したがって、このファンクションコールの実行後FIBを使用する場合には注意が必要です。

ファイル・サブディレクトリの移動

機能番号　4FH(_MOVE)

コール手順　DE　ドライブ・パス・ファイル ASCIIZ 文字列または FIB ポインタ
HL　新しいパス ASCIIZ 文字列

戻り値　A　エラー

解説　このファンクションはドライブ・パス・ファイル文字列あるいはFIBで指定されるファイルあるいはサブディレクトリを、HLで指される新しいパス文字列で指定されるディレクトリに移動します。新しいパス文字列中にはドライブ名があってはなりません。「CON」などのデバイス名が指定されるとエラーは返されませんが、そのデバイスは実際には移動されません。

ワイルドカード文字はどの文字列中でも使用できないため、このファンクションで移動できるファイルもしくはサブディレクトリはひとつだけです。ただしサブディレクトリが移動された場合、その下にあるすべてのファイルおよびサブディレクトリは、それに伴って移動されます。すでに移動先のディレクトリ中に指定の名前のエントリが存在すると、「.DUPF」エラーが返されて重複したファイル名が作成されることを防ぎます。サブディレ

クトリ中の「.」と「..」エントリは移動できず（「.DOT」エラー）、ディレクトリはファイルシステム中で孤立したループを生成してしまうことになるため、自分自身の下に移動できません（「.DIRE」エラー）。オープンされているファイルハンドルを持つファイルは移動できません（「.FOPEN」エラー）。

FIB がこのファンクションに渡されても、FIB の内部情報はファイルの新しい位置に対応するようには更新されません。そうしないと FIB を続く「次のエントリの検索」ファンクションコール（P.292参照）で使用できなくなります。しかし、FIB は移動されたファイルをもう参照していないということになりますので、「名前の変更」や「オープン」などの処理に、このファンクションに渡した FIB を使用してはいけません。

ファイル属性の獲得・セット

機能番号　50H(_ATTR)

コール手順
DE　ドライブ・パス・ファイル ASCIIZ 文字列または FIB ポインタ
A　0　属性の獲得
　　1　属性のセット
L　新しい属性バイト（A=1 の場合のみ）

戻り値
A　エラー
L　現在の属性バイト

解説

このファンクションはファイルやサブディレクトリの属性を変更するために使用されます。また現在の属性を知るために使用することができますが、これは「最初のエントリの検索」ファンクション（P.291参照）を使用するのがより一般的です。A=0 であるとファイルやサブディレクトリの現在の属性バイトが L レジスタに返されます。

A=1 の場合、属性バイトは L レジスタで指定された新しい値にセットされ、この新しい値も L レジスタ中に返されます。ファイルについてはシステム、不可視、読み出し専用、およびアーカイブビットを変更することができ、サブディレクトリについては不可視属性を変更できます。そのほかの属性ビットを変更しようとすると「.IATTR」エラーが返されます。FIB が渡されると、その中の属性バイトは新しい設定で更新されません。

このファンクションではワイルドカードは使用できず、属性をセットできるファイルおよびサブディレクトリはひとつだけです。ルートディレクトリには属性がないため、その属性を変更することはできません。ファイ

ルハンドルがオープンされているファイルの属性を変更することはできません（「.FOPEN」エラー）。一方「.」および「..」ディレクトリエントリの属性は変更することができます。「CON」などのデバイス名が指定されてもエラーは返されませんが、デバイスの属性は実際には変更されません。

ファイルの日付および時刻の獲得・セット

機能番号　51H(_FTIME)

コール手順
DE　ドライブ・パス・ファイル ASCIIZ 文字列または FIB ポインタ
A　0　日付と時刻の獲得
　　1　日付と時刻のセット
IX　新しい時刻の値（A=1 の場合のみ）
HL　新しい日付の値（A=1 の場合のみ）

戻り値
A　エラー
DE　ファイルの時刻の現在値
HL　ファイルの日付の現在値

解説

A=1 の場合、このファンクションはドライブ・パス・ファイル文字列あるいはFIBで指定されるファイルまたはサブディレクトリの最終修正時刻および日付をセットします。このファンクションではワイルドカード文字が使用できないため、日付と時刻を修正することができるファイルはひとつだけです。「CON」などのデバイス名が指定されてもエラーは返されませんが、デバイスの日付と時刻は実際には変更されません。

日付と時刻の形式はディレクトリエントリと FIB 中に保存されているものとまったく同一です（13章「ディスクファイルの構造」を参照）。日付や時刻についてのチェックは行われず、値はただセットされるだけです。FIB が渡されると、それにセットされた日付と時刻はこのファンクションでは更新されません。

A=0 の場合、現在の値がそのまま返されます。時刻の値を IX に渡しますが、結果は DE に返されるということに注意して下さい。オープンされているファイルハンドルがあるファイル（「.FOPEN」エラー）の日付と時刻は変更できません（読み出しは可）。

ファイルハンドルの削除

機能番号	52H(_HDELETE)
コール手順	B ファイルハンドル
戻り値	A エラー

解説

このファンクションは指定したファイルを削除し、そのファイルハンドルをクローズします。同一のファイルに対して別個にオープンされたファイルハンドルがある場合には、ファイルハンドルは削除できません（「.FOPEN」エラー）。ファイルハンドルの複製（「ファイルハンドルの複製」あるいは「子プロセスの起動」ファンクションで作成されるもの）が存在すると、これらの複製は使用できないようにマークされ、これらを使用しようとすると「.HDEAD」エラーとなります。

このファンクションのエラーの状態は、「ファイルあるいはサブディレクトリの削除」ファンクション（P.304参照）と同じです。エラーの状態（「.FILRO」や「.FOPEN」など）が存在しても、ファイルハンドルは常にクローズされます。

ファイルハンドルの名前の変更

機能番号	53H(_HRENAME)
コール手順	B ファイルハンドル HL 新しいファイル名ASCIIZ文字列
戻り値	A エラー

解説

このファンクションは指定のファイルハンドルに結合したファイルの名前を、HLで指される文字列中の新しい名前に変更します。ファイルがASCIIZ文字列やFIBではなくファイルハンドルで指定されるということを別にして、このファンクションは「ファイル名あるいはサブディレクトリ名の変更」ファンクション（P.305参照）と同じで、同一のエラー状態を持ちます。

ファイルハンドルはそのファイルに対してオープンされた別のファイルハンドルが存在する場合には名前を変更できません（「.FOPEN」エラー）が、このファイルハンドルのコピーが存在していても名前を変更でき、この場合コピーの名前も変更されます。ファイルハンドルの名前の変更はファイルポインタを変えませんが、それは暗黙の「確保」処理を実行します。

ファイルハンドルの移動

機能番号	54H(_HMOVE)
コール手順	B ファイルハンドル HL 新しいパス ASCIIZ 文字列
戻り値	A エラー

解説

このファンクションは指定のファイルハンドルに結合したファイルを、HLで指される新しいパス文字列で指定されるディレクトリへ移動します。ファイルがASCIIZ文字列やFIBではなくファイルハンドルで指定されるということを別にして、このファンクションは「ファイルあるいはサブディレクトリの移動」ファンクション（P.306参照）と同じで、同一のエラー条件を持ちます。

ファイルハンドルはそのファイルに対してオープンされた別のファイルハンドルが存在する場合には移動できません（「.FOPEN」エラー）が、このファイルハンドルのコピーが存在していても移動でき、この場合コピーも移動されます。ファイルハンドルの移動はファイルポインタを変えませんが、それは暗黙の「確保」処理を実行します。

ファイルハンドルの属性の獲得・セット

機能番号	55H(_HATTR)	
コール手順	B ファイルハンドル	
	A 0	属性の獲得
	1	属性のセット
	L 新しい属性バイト	A=1の場合のみ
戻り値	A エラー	
	L 現在の属性バイト	

解説

このファンクションは指定のファイルハンドルに結合したファイルの属性バイトを獲得または変更します。ファイルがASCIIZ文字列やFIBではなく、ファイルハンドルで指定されるということを別にして、このファンクションは「ファイル属性の獲得・セット」ファンクション（P.307参照）と同じで、同一のエラー状態を持ちます。

ファイルハンドルはそのファイルについてオープンされている別のファイルハンドルが存在する場合には、その属性を（獲得することはできますが）変更することはできません（「.FOPEN」エラー）。ファイルポインタは変更されませんが、暗黙の「確保」処理が実行されます。

ファイルハンドルの日付および時刻の獲得・セット

機能番号　56H(_HFTIME)

コール手順
B　ファイルハンドル
A　0　日付と時刻の獲得
　　1　日付と時刻のセット
IX　新しい時刻の値　A=1 の場合のみ
HL　新しい日付の値　A=1の場合のみ

戻り値
A　エラー
DE　ファイルの時刻の現在値
HL　ファイルの日付の現在値

解説
このファンクションは指定のファイルハンドルに結合したファイルの日付と時刻を獲得または変更します。ファイルが ASCIIZ 文字列や FIB ではなく、ファイルハンドルで指定されるということを別にして、このファンクションは「ファイル日付および時刻の獲得・セット」ファンクション(P.307参照)と同じで、同一のエラー状態を持ちます。

ファイルハンドルはそのファイルについてオープンされている別のファイルハンドルが存在する場合には、その日付と時刻を（獲得することはできますが）変更することはできません（「.FOPEN」エラー）。ファイルポインタは変更されませんが、暗黙の「確保」処理が実行されます。

ディスク転送アドレスの獲得

エラー処理

機能番号　57H(_GETDTA)

コール手順　なし

戻り値　DE　現在のディスク転送アドレス

解　説　このファンクションは現在のディスク転送アドレスを返します。このアドレスは「旧来の」CP/MスタイルのFCBファンクションやアブソリュートなセクタの読み出し・書き込みファンクションのために使用されます。

ベリファイフラグ設定の獲得

エラー処理

機能番号　58H(_GETVFY)

コール手順　なし

戻り値　B　00H　ベリファイ無効
　　　FFH　ベリファイ有効

解　説　このファンクションは「ベリファイフラグのセット/リセット」ファンクション(p.289参照)でセットすることのできるベリファイフラグの現在の状態を返します。

カレントディレクトリの獲得

機能番号　59H(_GETCD)

コール手順　B　ドライブ番号　0=カレント、1=A:など
DE　64バイトバッファへのポインタ

戻り値　A　エラー
DE　カレントパスで満たされる

解　説　このファンクションは指定のドライブのカレントディレクトリを表すASCIIZ文字列を獲得して、DEで指されるバッファ中に格納します。文字列にはドライブ名や前後の「¥」文字は含まれないので、ルートディレクトリはヌル文字列で表現されます。ドライブがアクセスされて、カレントディレクトリが実際に現在のディスク上に存在するということが確認されるため、もし存在しない場合にはカレントディレクトリはルートにセットし直され、ヌル文字列が返されます。

カレントディレクトリの変更

機能番号	5AH(_CHDIR)
コール手順	DE ドライブ・パス・ファイル ASCIIZ 文字列
戻り値	A エラー

解説

ドライブ・パス・ファイル文字列はファイルではなくディレクトリを指定しなければなりません。そのドライブのカレントディレクトリは指定されたディレクトリに変更されます。指定のディレクトリが存在しない場合現在の設定は変更されず、「.NODIR」エラーが返されます。

パス名の解析

機能番号	5BH(_PARSE)
コール手順	B ボリューム名フラグ（ビット 4） DE 解析する ASCIIZ 文字列
戻り値	A エラー DE 終了文字へのポインタ HL 最後の項目の先頭へのポインタ B 解析フラグ C 論理ドライブ番号（1=A:など）

解説

このファンクションは純粋な文字列操作ファンクションで、ディスクをまったくアクセスせず、またユーザーの文字列にいかなる変更も加えません。これは外部プログラムがコマンド行を解析する手助けのために用意されています。

ボリューム名フラグ（B レジスタのビット 4。これは属性バイトにおけるボリューム名と同じビット位置にある）がクリアされているとき文字列を、「ドライブ・パス・ファイル」文字列として、セットされているとき「ドライブ・ボリューム」文字列として解析します。

DE に返されるポインタはパス名文字列中で有効でない最初の文字を指し、文字列の終わりのヌルのこともあります。パス名文字列の構文の詳細や有効な文字のリストについては5.1「この章の表記法」を参照して下さい。

HL に返されるポインタは文字列の最後の項目（「ファイル名」部分）の最初の文字を指します。例えば、渡された文字列が「A:¥XYZ¥P.Q /F」

なっば、DE は/F の前の空白文字を指し、HL は「P」を指します。解析された文字列が「¥」文字で終わっていたり、あるいはヌル（ドライブ名を別として）であると、これらの場合には「最後の項目」は存在しないため、HL は DE と同一の文字を指します。この特殊な場合には、このファンクションを使用するすべてのプログラムでなんらかの特別な処理がふつっ必要となります。

C レジスタに返されるドライブ番号は文字列中で指定される論理ドライブです。文字列がドライブ文字で始まらないと、カレントドライブが暗黙で指定されたということなので、C レジスタはカレントドライブ番号を保持しています。C レジスタは0になることはありません。

B レジスタに返される解析フラグは文字列に関する様々な情報を示します。ボリューム名では、ビット 1、4、5、6、および7は常にクリアされます。ファイル名では、ビット 3～7 は文字列の最後の項目（「ファイル名」部分）に関連します。ビットの割り当てを次に示します。

ビット	意味
0	ドライブ名以外の文字が解析されたときにセット
1	ディレクトリパスが指定されたときにセット
2	ドライブ名が指定されたときにセット
3	最後の項目で主ファイル名が指定されたときにセット
4	最後の項目でファイル名拡張子が指定されたときにセット
5	最後の項目にワイルドカードがあるときにセット
6	最後の項目が「.」あるいは「..」のときにセット
7	最後の項目が「..」のときにセット

ファイル名の解析

機能番号	5CH(_PFILE)
コール手順	DE 解析のための ASCIIZ 文字列 HL 11バイトバッファへのポインタ
戻り値	A エラー（常に0） DE 終了文字へのポインタ HL 保存、バッファが使用 B 解析フラグ

解説

このファンクションは純粋な文字列操作ファンクションであり、まったくディスクにアクセスせず、文字列に修正を加えることはまったくありません。このファンクションは主に外部プログラムがフォーマットされた形式でファイル名を出力する手助けになる為に用意されています。ASCIIZ文字列は単一のファイル名の項目として解析され、ファイル名はユーザーの11バイトのバッファに拡張された形式でセットされ、ファイル名および拡張子の余白には空白が詰められます。

Bレジスタに返される解析フラグは上記の「パス名の解析」ファンクション（P.313参照）と同じですが、ビット0、1、および2は常にクリアされています。例え文字列中に有効なファイル名がなかったとしてもユーザーのバッファは常に満たされ、その場合にはバッファは空白で満たされます。「*」文字は適当な数の「?」に展開されます。ファイル名あるいはファイル名拡張子が長すぎると余分な文字は無視されます。

DEレジスタに返されるポインタはファイル名の一部ではなかった文字列中の最初の文字を指しますが、これは文字列の終わりにあるヌルである場合があります。この文字は有効なファイル名の文字であることはありません（有効なファイル名の文字の詳細については5.1「この章の表記法」を参照）。

文字の検査

エラー処理

機能番号　5DH(.CHKCHR)

コール手順　D　文字フラグ
E　検査する文字

戻り値　A　0(エラーを返すことはない)
D　更新された文字フラグ
E　検査された(大文字にされた)文字

解説

このファンクションは言語設定とは独立して文字を大文字にしたり、2バイトの文字やファイル名の操作をしたりするのに役立ちます。文字フラグのビット割り当てを次に示します。

ビット	意味
0	大文字にしないときにセット
1	2バイト文字の第1バイトのときにセット
2	2バイト文字の第2バイトのときにセット
3	セット＝ボリューム名（ファイル名ではない）
4	セット＝有効なファイル・ボリューム名の文字
5〜7	システム予約（常にクリア）

ビット0は大文字化を制御するために使用します。これがクリアされると、文字はそのマシンの言語設定にしたがって大文字にされます。このビットがセットされていると、返される文字は常に渡された文字と同一になります。

2つの2バイト文字のフラグ（ビット1と2）は、文字列の最初の文字をチェックするときに両方ともクリアすることができ、返された設定は、続く文字のためにこのファンクションに直接渡すことができます。これらのフラグで、2バイト文字を含む可能性のある文字列を逆戻りする場合には注意が必要です。

ビット4は文字がファイル名やボリューム名の終了文字であると、セットされてリターンします。ビット3は単純に、ファイル名の検査か、あるいはボリューム名の文字の検査かを決定するのに使用されます（文字セットが異なるため）。2バイト文字（どちらのバイトも）はボリュームあるいはファイル名の終了文字として見なされることはありません。

完全なパス文字列の獲得

機能番号　5EH(_WPATH)

コール手順　DE　64バイトバッファへのポインタ

戻り値　A　エラー
DE　完全なパス文字列で満たされる
HL　最後の項目のはじめへのポインタ

解説　このファンクションは内部バッファからASCIIZパス文字列をユーザーのバッファへ単純にコピーします。文字列は、前に実行された「最初のエントリの検索」(P.291参照) あるいは「新しいエントリの検索」(P.293参照) で見つけられたファイルやサブディレクトリのルートディレクトリからの完全なパスとファイル名を表します。返された文字列はドライブや最

初の「¥」文字を含みません。HL レジスタは「パスの解析」ファンクション（P.313参照）と同様、文字列上の最後の項目の最初の文字を指しています。

「最初のエントリの検索」や「新しいエントリの検索」ファンクションコールでDE が ASCIIZ 文字列を指して実行された場合、続く「完全なパス名の獲得」ファンクションコール（P.316参照）は「検索」ファンクションコールで返される FIB に関連するサブディレクトリやファイルを表す文字列を返します。これがサブディレクトリであると、FIB はレジスタ DE で別の「最初のエントリの検索」ファンクションコールに渡すことができこのサブディレクトリ中でファイルが検索されます。この場合、新たに検索されるファイルはすでに内部で存在する完全なパス文字列に加えられ、続く「完全なパス文字列の獲得」ファンクションコールは検索されたファイルについて正しい完全なパス文字列を返します。

多くのファンクションコールで内部の完全なパス文字列が変更され、大抵の場合正しくないので、このファンクションを使用する場合には十分な注意が必要です。「完全なパスの獲得」ファンクションコールは、それが関係する「最初のエントリの検索」や「新しいエントリの検索」ファンクションの直後に行うべきです。

ディスクバッファのフラッシュ

機能番号	5FH(_FLUSH)		
コール手順	B	ドライブ番号	(0=カレント、FFH=すべて)
	D	00H	フラッシュのみ
		FFH	フラッシュして無効にする
戻り値	A	エラー	

解　説

このファンクションは指定のドライブ、あるいはB=FFHの場合にはすべてのドライブについて、まだ書き出されていないすべてのディスクバッファをフラッシュします。D レジスタが FFH であると、そのドライブのすべてのバッファも無効になります。

子プロセスの起動

機能番号	60H(_FORK)
コール手順	なし
戻り値	A エラー B 親プロセスのプロセスID

解説

このファンクションはシステムに対して子プロセスが起動されようとしていることを報告します。一般的には、これは実行されようとする新しいプログラムやサブコマンドです。例えば、COMMAND2.COMはすべてのコマンドや外部プログラムを実行する前に、「子プロセスの起動」ファンクションコールを実行します。

新しいファイルハンドルのセットが作成され、「継承アクセスモードビット」をセットしてオープンされたすべての現在のファイルハンドル（「ファイルハンドルのオープン」ファンクション（P.295参照）は新しいファイルハンドルのセットにコピーされます。「継承ビット」をクリアしてオープンされたすべてのファイルハンドルはコピーされないため、子プロセスでは利用できません。標準ファイルハンドル（00H〜05H）は継承できるため、これらはコピーされます。

新しいプロセスIDが子プロセスのために割り当てられ、親プロセスのプロセスIDが返され、後で「親プロセスに戻る」ファンクションコール（P.318参照）で親プロセスへ戻れるようにします。ファイルハンドルを複製するのにメモリが足りない場合には「.NORAM」エラーが返されます。

子プロセスはオリジナルではなくて以前のファイルハンドルのコピーを持つため、コピーのひとつがクローズされても元のものはオープンされたままです。したがって、例えば子プロセスが標準出力のファイルハンドル（ファイルハンドル番号1）をクローズし、それを新しいファイルに再オープンすると、「親プロセスに戻る」ファンクションが実行されて親プロセスに戻ったとき、元の標準出力チャンネルはまだ有効です。

親プロセスに戻る

機能番号	61H(_JOIN)
コール手順	B 親のプロセスIDまたは0

戻り値

A　エラー
B　子プロセスからの1次エラーコード
C　子プロセスからの2次エラーコード

解説

このファンクションは指定の親プロセスに戻り、子プロセスが終了したエラーコードをBレジスタに入れ、子プロセスからの2次エラーコードをCレジスタに入れてリターンします。親と子プロセス間の関係は厳密に1対1ですが、このファンクションは適切なプロセスIDを与えることによって、いくつかのレベルを飛び越えて戻ることができます。プロセスIDが不正であると「.IPROC」エラーが返されます。

子プロセスのファイルハンドルのセットは自動的にクローズされ、親プロセスのファイルハンドルのセットが再びアクティブになります。子プロセスが割り当てていたすべてのユーザーRAMセグメントも解放されます。

このファンクションに渡されたプロセスIDが0であると、部分的なシステムの再初期化が実行されます。すべてのファイルハンドルがクローズされ、標準入出力が再オープンされ、すべてのユーザーセグメントが解放されます。この後ではコマンドインタープリタは正しい状態ではないためユーザープログラムはコマンドインタープリタに戻ろうとするならばこれを行ってはなりません。

このファンクションは実際にファイルハンドルをクローズする前に（つまりディスクにアクセスする前に）、メモリの解放とプロセスIDの調整を行うように、慎重な配慮がなされています。これによって、ディスクエラーが起こって中断されても、処理は成功しているということが保証されます。しかし「join 0」がディスクエラーを起こしてアボートすると、デフォルトファイルハンドルの再初期化は実行されません。この場合には別の「join 0」ファンクションコールを実行しなければならず、これはディスクをアクセスしようとしないため（すべてのファイルはクローズされているため）成功します。

このファンクションコールが0F37DHを通して実行されると　BおよびCレジスタはエラーコードを返さないことに注意して下さい。これは、プログラムの終了や中断の処理はアプリケーションプログラムによって実行されなければならないからです。エラーコードはアボートルーチンに渡されており、そこにあるプログラムが必要ならばエラーコードを記憶していなければなりません。1次および2次エラーコードの意味については「エラーコードを伴った終了」ファンクション（P.320）も参照して下さい。

エラーコードを伴った終了

機能番号　62H(_TERM)

コール手順　B　終了のエラーコード

戻り値　なし

解説　このファンクションは指定されたエラーコードでプログラムを終了させますが、このコードはエラーなしを示す0であってもかまいません。このファンクションコールは、ユーザーのアボートルーチンが戻るように設定されていない限り、呼び出し側に戻ることはありません（P.320「アボート終了ルーチンの定義」を参照)。このファンクションの操作は0005Hを通って MSX DOSの環境からコールされたのか、あるいは0F37DHを通って Disk BASICの環境からコールされたのかによって異なります。

0005Hを通ってコールされた場合、ユーザーのアボートルーチンが「アボート終了ルーチンの定義」ファンクションによって定義されているとそれは指定のエラーコード（および0の2次エラーコード）を伴ってコールされます。このルーチンがリターンするか、またはユーザーのアボートルーチンが定義されていない場合、制御はアドレス0000Hへのジャンプを経由して外部プログラムをロードしたルーチンへ戻されます。これはほとんど常にコマンドインタープリタですが、場合によっては別の外部プログラムであることもあります。エラーコードはシステムによって記憶され、実行される次の「親プロセスに戻る」ファンクション（P.318参照）がこのエラーコードを返します。コマンドインタープリタは20H～FFHの間のすべてのコードについてエラーメッセージを表示しますが、それ以下のエラーについてはメッセージを表示しません。

このファンクションが0F37DHを通してDiskBASICの環境からコールされると、制御は位置「BREAKVECT」のアボートベクタへ渡されます。この環境では、別に定義されたユーザーのアボートルーチンはなく「join」はエラーコードを返さないため、エラーコードは「BREAKVECT」にあるコードによって記憶されなければなりません。

アボート終了ルーチンの定義

機能番号　63H(_DEFAB)

コール手順　DE　アボート終了ルーチンのアドレス
0000Hで定義解除

戻り値　A　0　エラーを生成しない

解説　このファンクションはMSX DOSの環境で0005H番地を通してコールされたときに利用できます。Disk-BASICの環境で0F37DHを通してはコールできません。

DEレジスタが0であると、前に定義したアボートルーチンが定義を解除され、そうでないと新しいものが定義されます。0000H番地に直接ジャンプするのではなく、外部プログラムがなんらかの理由で終了しようとしているときには必ずシステムによってアボートルーチンがコールされます。MSX DOS2のために書かれたプログラムでは、0000H番地へのジャンプではなくて、「エラーコードを伴った終了」ファンクションコール（P.320参照）で終了しなければなりません。

ユーザーのアボートルーチンはユーザーのスタックをアクティブにしファンクションコールが行われたときと同じようにIX、IYと裏レジスタセットをセットし、TPA全体をページングして起動されます。終了エラーコードはAレジスタ、2次エラーコードはBレジスタでルーチンに渡され、ルーチンが「RET」を実行すると、AおよびBレジスタに返された値が「親プロセスに戻る」ファンクション（P.318参照）で返されるべきエラーコードとしてストアされ、通常コマンドインタープリタによって出力されます。あるいは、ルーチンはリターンせずに外部プログラム中のなんらかのウォームスタートコードにジャンプしてもかまいません。システムは完全に安定した状態にあって、どのようなファンクションコールも受け付けることができます。

Aレジスタに入れてルーチンに渡される1次エラーコードは、プログラム自身が「エラーコードを伴った終了」ファンクション（P.320参照）に渡したコード（0でもよい）になります（これが終了の理由ならば）。

[CTRL]+[C]あるいは[CTRL]+[STOP]が検知された場合（「.CTRLC」または「.STOP」エラー）、ディスクエラーがアボートされた場合（「.ABORT」エラー）、あるいはMSX DOSのファンクションコール01H～0BHを通してアクセスしている標準入力あるいは出力のチャンネルのひとつでエラーが起こった場合（「.INERR」または「.OUTERR」）にもルーチンはコールされます。「.ABORT」、「.INERR」、および「.OUTERR」エラーはなんらかの他のエラーの結果としてシステムによって生成されます。例えば、「.ABORT」エラーは「.NRDY」エラーによって起こる可能性があり、「.INERR」エラーは「.EOF」エラーで起こる可能性があります。これらの場合には、元のエラーコード（「.NRDY」や「.EOF」）は2次エラーコードとしてBレジスタでアボートルーチンに渡されます。その他のすべてのエラーは2次エラーコードを持たず、Bレジスタは0です。

アボートルーチンが単純にリターンせず、「POP HL : RET」(または同等のコード) を実行すると、制御はエラーが起こったMSX DOSコールまたはBIOSコールのすぐ次の命令に渡ります。これは「ディスクエラー処理ルーチンの定義」ファンクション (P.322参照) とともに使用すると有用で、ディスクエラーが起こったときに現在のMSX-DOSコールをアボートすることが可能になります。

ディスクエラー処理ルーチンの定義

機能番号　64H(_DEFER)

コール手順　DE ディスクエラールーチンのアドレス
0000Hで定義解除

戻り値　A 0 エラーを生成しない

解説

このファンクションはディスクエラーが起こった場合にコールされるユーザーのルーチンのアドレスを指定します。TPA全体をページングしてルーチンは起動されますが、ページ3のシステムスタックがアクティブになり、MSX DOSのファンクションコールが実行されたときのレジスタは保存されません。

エラールーチンはMSX-DOSのコールを実行できますが、エラーの再帰を避けるように十分注意しなければなりません。17.2「ファンクション一覧」はユーザーのエラールーチンからどのファンクションコールが安全に実行できるかを示しています。標準入出力チャンネルがリダイレクトされている場合には、このルーチンはリダイレクション状態を一時的に無効にした状態でコールされます。これについての詳細は、「リダイレクション状態の獲得・セット」ファンクション (P.329) を参照して下さい。

ルーチン自体のパラメータと結果の仕様を下に示します。IX、IYおよび裏レジスタセットを含むすべてのレジスタは破壊できますが、ページングとスタックは保存しなければなりません。ルーチンはシステムに返らねばならず、外部プログラムを続行するために別のところにジャンプしてはなりません。これを行うには、A=1(「アボート」)を返すことで、ユーザーのアボートルーチンに制御を渡して、アボートルーチン内で必要な処理を行うようにします。

パラメータ　A ＝エラーの原因となったエラーコード
B ＝物理ドライブ
C ＝b0 － 書き込みでセット
＝b1 － 無視の処理が望ましくないときセット
＝b2 － オートアボートを指示するときセット
＝b3 － セクタ番号が有効のときセット
DE＝セクタ番号（C の b3 がセットされているとき）
結果　A ＝0 ＝ システムエラールーチンのコール
＝1 ＝ アボート
＝2 ＝ 再試行
＝3 ＝ 無視

直前のエラーコードの獲得

エラー処理

機能番号　65H(_ERROR)

コール手順　なし

戻り値　A　0
B　直前のファンクションからのエラーコード

解説　このファンクションによって、ユーザープログラムは前の MSX-DOS ファンクションコールが失敗した原因のエラーコードを見つけることができます。これはエラーコードを返さない古い CP/M 互換のファンクションで使用するためのものです。例えば、「ファイル FCBの作成」ファンクションがA＝0FFH を返すと、失敗の理由は数多くある可能性がありますがこのファンクションコールを使用すると、例えば「.DRFUL」や「.SYSX」など適切なものを返します。

エラーコードの説明

エラー処理

機能番号　66H(_EXPLAIN)

コール手順　B　説明すべきエラーコード
DE　64 バイトの文字列バッファへのポインタ

戻り値	
A	0
B	0あるいは変更なし
DE	エラーメッセージが入る

解説

このファンクションを使用すると、ユーザープログラムでMSX-DOSのファンクションから返される特定のエラーコードについてのASCIIZ説明文字列を得ることができます。もしエラーが旧来のファンクションのものであれば、「直前のエラーコードを得る」ファンクションを最初に呼んで本当のエラーコードを得なければならず、それからこのファンクションをコールして説明文字列を得ることができます。

16章「エラー」にすべての現在定義されているエラーコードとそれに対するメッセージのリストがあります。システムの外国向けのバージョンではもちろん異なったメッセージとなります。エラーコードが実際に内部の説明文字列を持っていると、この文字列が返されてBレジスタが0にセットされます。説明文字列がないと、「システムエラー 194」あるいは「ユーザエラー 45」という形式の文字列が返され、Bレジスタは変更されません。(システムエラーは40H～FFHの範囲のもので、ユーザエラーは00H～3FHのもの)。

ディスクのフォーマット

機能番号　67H(_FORMAT)

コール手順

B	ドライブ番号(0=カレント、1=A:)	
A	00H	選択文字列を返す
	01H～09H	この選択をフォーマットする
	0AH～FDH	不正
	FEH, FFH	ブートセクタの更新
HL	バッファへのポインタ(A=1～9の場合)	
DE	バッファのサイズ(A=1～9の場合)	

戻り値

A	エラー
B	選択文字列のスロット(エントリでA=0の場合のみ)
HL	選択文字列のアドレス(エントリでA=0の場合のみ)

解説

このファンクションはディスクをフォーマットするために使用され、アプリケーションプログラムで必要な場合には使用することもできますが、事実上FORMATコマンドのためにだけ提供されています。Aレジスタに渡されたコードによって選択される3つの異なったオプションがあります。

A=0の場合、BとHLレジスタはフォーマットの種類を選択するASCIIZ文字列のスロット番号とアドレスをそれぞれ返します。このディスクがフォーマットできない（例えばRAMディスク）と「.IFORM」エラーが返されます。文字列は「RDSLT」ルーチンを使用して読み込まれ、画面に表示されて、それから「?」プロンプトが表示されます。ユーザーはそこで「1」～「9」の選択を指定し、この選択が「フォーマット」ファンクションに適切な警告プロンプトの後で渡され、実際のディスクのフォーマットが実行されます。A=0でHLに0が返される場合があり、これはフォーマットが1種類しかなくプロンプトの必要が無いことを意味します。これらの処理はディスクドライバごとに異なるため、それぞれの選択がどのようなディスクフォーマットを参照するのかを知る方法はありません。

A=01H～09Hの場合、これはフォーマットの選択として解釈され、それ以上のプロンプトなしでディスクは指定のドライブでフォーマットされます。HLとDEレジスタはディスクドライバによって使用されるバッファ領域を指定しなければなりません。このバッファがどの位の大きさであるべきか知る方法はないので、可能な限り大きくするのが最良の方法です。バッファがページ境界をまたぐと、このファンクションはディスクドライバに渡すために、ひとつのページ中の最大の部分を選択します。ほとんどのディスクドライバはこのバッファをまったく使用しません。

A=FFHの場合、ディスクは実際にはフォーマットされませんが、ディスクに新しいブートセクタを書き込み、MSX-DOS2ディスクにします。これは以前のMSX-DOS1のディスクをボリュームIDを持つように更新して、それによってMSX DOS2で可能な完全なディスク検査とファイルの復活を可能にするためです。A=FEHはA=FFHとほぼ同様ですが、ディスクパラメータのみを正しく更新し、ボリュームIDでブートプログラムを重ね書きすることはありません。ディスク上に不正なブートセクタを置くいくつかのMSX-DOS1のインプリメンテーションが存在するため、それらのディスクはこのファンクションで訂正されるまでMSX-DOS2では使用することができません。

「ブートセクタの更新」は主にFIXDISKコマンドのために設けられていますが、それが有用な場合には他のプログラムで使用することができます。それを使用する場合には、「フォーマットの選択の獲得」(A=0) が最初に実行されなければならず、これがエラー（通常「.IFORM」）を返すとこれがフォーマットできないドライブで、ディスクはこのファンクションで損傷されることがあるため、処理はアボートされなければなりません。

RAMディスクの作成あるいは消去

機能番号 68H(_RAMD)

コール手順 B 00H RAMディスクを消去
1~FEH 新しいRAMディスクの作成
FFH RAMディスクのサイズを返す

戻り値 A エラー
B RAMディスクのサイズ

解説 Bレジスタ=0FFHの場合、このルーチンは、現在のRAMディスクに割り当てられている16KRAMセグメントの数を返します。値0は現在定義されているRAMディスクが存在しないことを示します。B=0の場合現在のRAMディスクが消去され、それが保持していたすべてのデータは失われ、RAMディスクがなかった場合でもエラーが返されません。

また、Bが01H~FEHの範囲にある場合、このファンクションはBレジスタに指定された16Kセグメントの数を使用して新しいRAMディスクを作成しようとします。RAMディスクがすでにある場合(「.RAMDX」)あるいは未使用のセグメントがひとつもない場合(「.NORAM」)、エラーが返されます。指定のサイズのRAMディスクを作成するのに十分な未使用RAMセグメントがないと可能な最大のものが作成されます。この場合エラーは返されません。

すべての場合において、RAMディスクのサイズがセグメント数としてBレジスタに返されます。RAMのいくらかはファイルアロケーションテーブルとルートディレクトリに使われるため、DIRやCHKDSKコマンドで示されるRAMディスクのサイズは使用されるRAMの総計よりもやや小さくなります。システムの他のドライブ数に関わらず、RAMディスクは常にドライブ文字「H:」としてアクセスされます。

セクタバッファの割り付け

機能番号 69H(_BUFFER)

コール手順 B 0 バッファ数を返す
あるいは要求するバッファ数

戻り値 A エラー
B バッファの現在の数

解　説

B=0の場合、このファンクションは現在割り当てられているセクタバッファの数を返します。B≠0であると、このファンクションはこの数をセクタバッファ数として使用しようとします（常に最低でも2は必要です）。それが要求されただけのバッファを割り当てられないと、可能な限り割り当てを行い、Bレジスタにその数を返しますが、エラーは返しません。セクタバッファの数は増やすことも減らすこともできます。

セクタバッファは通常の64Kの外の16KRAMセグメントに割り当てられるため、バッファ数はTPAのサイズを減少させません。しかしバッファが多ければより多くのFATとディレクトリセクタを常駐させておくことができるため、バッファ数は効率に影響します。バッファの最大数は20くらいになります。

論理ドライブの割り当て

エラー処理

機能番号

6AH(_ASSIGN)

コール手順

B　論理ドライブ番号（1=A:など）
D　物理ドライブ番号（1=A:など）

戻り値

A　エラー
D　物理ドライブ番号（1=A:など）

解　説

このファンクションは論理−物理ドライブの割り当て機能を制御します。これは主としてASSIGNコマンドのために提供されていますが、ユーザープログラムで使用して論理ドライブ番号を物理ドライブ番号に変換することもできます。

BとDがどちらも0でないと、新しい割り当てがセットアップされます。Bレジスタが0でなく、Dレジスタが0であると、Bで指定した論理ドライブについての割り当てがキャンセルされます。BとDレジスタがどちらも0であると、すべての割り当てがキャンセルされます。Bレジスタが0でなく、DレジスタがFFHであると、Bレジスタで指定した論理ドライブについての現在の割り当てが、Dレジスタに返されます。

ファンクションコールへの文字列中のドライブ名とパラメータとしてのドライブ名を含め、様々なファンクションコールで使用されるすべてのドライブは論理ドライブです。しかし、ディスクエラールーチンに渡されるドライブ番号は物理番号であるため、ASSIGNコマンドを使用していると対応する論理ドライブとは異なる場合があります。

環境変数の獲得

エラー処理

機能番号 6BH(_GENV)

コール手順
HL ASCIIZ名前文字列へのポインタ
DE 値のためのバッファへのポインタ
B バッファサイズ

戻り値
A エラー
DE 保存される、A=0の場合にバッファが満たされる

解説
このファンクションはHLレジスタに渡された環境変数名の現在の値を獲得します。環境変数の大文字、小文字の違いは無視されます。名前の文字列が不正であると「.IENV」エラーが返されます。その名前の環境変数がないとバッファにヌル文字列が返されます。その名前の変数があるとその値の文字列がバッファにコピーされます。バッファが小さくて入らないと、値の文字列は最後のヌルを付けずに切り捨てられ、「.ELONG」エラーが返されます。値の文字列は255文字よりも長くできない（最後のヌルも含む）ため、255バイトのバッファは常に十分な大きさを持つことになります。

環境変数のセット

エラー処理

機能番号 6CH(_SENV)

コール手順
HL ASCIIZ名前文字列へのポインタ
DE ASCIIZ値文字列へのポインタ

戻り値
A エラー

解説
このファンクションは新しい環境変数をセットします。環境変数の大文字、小文字の違いは無視されます。名前文字列が不正であると「.IENV」エラーが返され、有効ならば値文字列がチェックされ、それが255文字よりも長い場合には「.ELONG」エラーが返され、また、新しい変数を格納するために十分なメモリがない場合には「.NORAM」エラーが返されます。すべてが有効ならば同じ名前の古い変数が削除され、新しい変数が環境リストの最初に追加されます。値文字列がヌルだと環境変数は除去されます。

環境変数の検索

エラー処理

機能番号　6DH(_FENV)

コール手順
DE　環境変数番号
HL　名前文字列のバッファへのポインタ
B　バッファサイズ

戻り値
A　エラー
HL　保存され、バッファが満たされる

解説

このファンクションはどのような環境変数が現在セットされているかを知るために使用されます。DE レジスタの変数番号は、リスト中のどの変数を検索するか（最初の変数がDE=1に対応）を示します。変数番号<DE>が存在すると、この変数の名前文字列がHLによって指されるバッファ中にコピーされます。バッファが小さすぎる場合には、名前は最後のヌルなしで切り捨てられ、「.ELONG」エラーが返されます。255 バイトのバッファは、常に十分な大きさを持ちます。変数番号<DE>が存在しないと、ヌル文字列が返されます（変数はヌル名文字列を持つことはないため）。

ディスク検査ステータスの獲得・セット

エラー処理

機能番号　6EH(_DSKCHK)

コール手順
A　00H　ディスク検査ステータスの獲得
　　01H　ディスク検査ステータスのセット
B　00H　有効（A=01H の場合のみ）
　　FFH　無効（A=01H の場合のみ）

戻り値
A　エラー
B　現在のディスク検査設定

解説

A=0 の場合、ディスク検査変数の現在の値がBレジスタに返されます。A=01H の場合、変数は B レジスタの値にセットされます。00H という値はディスク検査を有効にすることを意味し、0 でない値はそれが無効であることを示します。デフォルトの状態は有効です。

ディスク検査変数はファイルハンドル、FIB あるいは FCB がアクセスされるたびに、ディスクが変更されたかどうかを見るためにディスクのブー

トセクタをシステムが再チェックするかどうかを制御します。それが有効であると、処理の最中にディスクを交換することによって、間違ったディスクに誤ってアクセスすることができなくなりますが、有効でない場合にはディスクを破壊する場合もあります。ディスクインタフェースのタイプによって異なりますが、この機能を有効にする場合に若干余分なオーバーヘッドがある場合があります。しかし、ほとんどのタイプのディスク（ディスク交換を検知するためのハードウェアを持つもの）では処理時間に変わりはなく、これによって安全性が確保されます。

MSX-DOSのバージョン番号の獲得　エラー処理

機能番号　6FH(_DOSVER)

コール手順　なし

戻り値
A　エラー（常に0）
BC　MSX-DOSのカーネルバージョン
DE　MSXDOS2.SYSのバージョン番号

解説　このファンクションによって、プログラムは実行されているMSX-DOSのバージョンを判断することができます。2つのバージョン番号が返されBCレジスタにROM中のMSX-DOSのカーネルバージョンが、DEレジスタにMSXDOS2.SYSシステムファイルのバージョンが入ります。これらのバージョン番号のどちらも上位バイトに主バージョン番号、下位バイトに2桁のバージョン番号がBCD値で入っています。例えばバージョン2.34のシステムが存在したとすると、これは0234Hとして表現されます。

MSX-DOS1との互換性のために、次の手順がこのファンクションを使用するためには必要です。まず、エラーが存在する（A<>0）と、これはMSX DOSではありません。次にBレジスタに注目します。これが2以下であると、システムは2.00より以前のもので、CとDEレジスタは不定となります。Bレジスタが2以上の場合、BCとDEレジスタは上記のように使用することができます。通常この手順の後、チェックされなければならないバージョン番号は、DEレジスタ中のMSXDOS2.SYSのバージョンです。

リダイレクションの状態の獲得・セット

エラー処理

機能番号 70H(_REDIR)

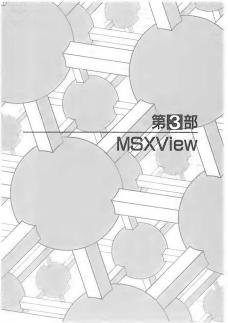
コール手順
A 00H リダイレクション状態の獲得
01H リダイレクション状態のセット
B 新しい状態
b0 標準入力
b1 標準出力

戻り値
A エラー
B コマンド以前のリダイレクションの状態
b0 セット 入力がリダイレクトされている
b1 セット 出力がリダイレクトされている

解説

このファンクションは主にディスクエラールーチンや、リダイレクションに関係なくコンソールに常に出力されなければならないその他の文字I/Oのために提供されています。CP/M文字ファンクション（ファンクション01H〜0BH）が使用されると、それらは通常コンソールを参照します。しかし、標準入力あるいは標準出力ファイルハンドル（ファイルハンドル0および1）がクローズされてディスクファイルに再オープンされていると、CP/M文字ファンクションもディスクファイルへ出力されます。しかしディスクエラー出力などのある種の出力はそれに関係なく常に画面に出力する必要があります。

このファンクションによって、このようなリダイレクションをこのファンクションをA=1およびB=0としてコールすることによって一時的に無効にすることができます。このファンクションをコールすることで、以降のCP/Mのコンソール I/Oがすべてコンソールに出力されることが保証され、また、直前の設定が返されるためこれを後でリストアすることができます。このようにリダイレクションの状態を変更すると、システムは幾分不安定な状態となり、多くのファンクションコールがこのファンクションを無効にしてリダイレクションをその実際の状態にリセットします。通常、「オープン」、「クローズ」、「複製」などといった、ファイルハントルを操作するすべてのファンクションコールはリダイレクションの状態をリセットします。したがってこのファンクションの効果は完全に一時的なものです。

第3部 MSXView

1章

MSXViewとは

MSXView とは、ユーザーインターフェイスに優れたアプリケーションプログラムを容易に開発するためのグラフィカルユーザーインターフェイスシステムです。MSXView に対応することで、MSX turbo R 用のアプリケーションプログラムの開発の作業量を大幅に減らすことができます。

1.1 開発者にとっての MSXView

従来 MSX turbo R では、それぞれのアプリケーションプログラムがポインティングデバイスの管理を行ったり、かな漢字変換システムや漢字フォントを内蔵していました。

しかし、MSXView の環境でアプリケーションプログラムを作成することにより、これらのシステムリソース（システム資源）をおのおののアプリケーションプログラムが管理する必要がないため、従来と比較してアプリケーション開発作業が大幅に緩和されます。

さらに、アプリケーションを MSXView の規定にしたがって作成すれば、将来に渡りさまざまな恩恵が受けられることになります。ポインティングデバイスの管理、かな漢字変換、デスクアクセサリ、各種フォント資源（漢字、デザイン文字など）などのシステム資源について、ハードウェアに変更があっても、MSXView がその違いを吸収するので アプリケーションはほとんど気にすることなく開発に集中できます。

1.2 ユーザーにとっての MSXView

MSXView を用いると、開発者側ばかりではなく、ユーザーにとっても大きなメリットを生み出します。

まず第一に、MSXView の環境下で作成されたアプリケーションは、操作性が統一されます。例えば、日本語入力の方法や起動、終了、セーブ、ロードなどの方法がすべて統一されたマナーで実行できます。このため、ユーザーが新しいアプリケーションの使用方法をマスターすることが簡単になります。これは、MSX turbo R で実用アプリケーションを普及するために、非常に重要なことです。

第二に、アプリケーション間でのデータの受渡しを円滑に行えます。これにより、あるアプリケーションで作成したデータを別のアプリケーションで利用することができ、一度コンピュータに入力したデータが無駄になりません。

2章
MSXViewファンクションの使い方

MSXView のルーチン群を使用するときは、8 番地にある MSXView ファンクションコールエントリを RST 命令でコールします。

通常のアプリケーション開発では、C 言語を使用することを推奨しています（C コンパイラは LSI-C version-2.0 以降のもの）。プログラムの中で処理スピードが問題になるような部分については、MSXView 内にアセンブラで記述されたルーチンがあるので、ほとんどのアプリケーションは、C 言語だけで開発することができます。例えば、VSHELLや ViewDRAW などでは、アセンブラはまったく使われていません。

C 言語を使用する通常のアプリケーション開発用には、MSXView ファンクションを、通常の関数と同様に記述できるようなマクロ定義を行うヘッダーファイル<function.h>が用意されています。このヘッダーファイルが、関数として記述された MSXView コールを 8 番地コールの RST 命令に展開するので、アプリケーション開発においては、MSXView の 8 番地コールの詳細について知る必要はありません。

現状では MSX 上で MSX-C を用いて MSXView のアプリケーションを開発することはできません。MSX 上では M-80、L-80 を使用して、アセンブラで開発を行います。アセンブラから MSXView ファンクションコールを使用するための定義として、ヘッダーファイル<function.inc>が用意されています。なお、MSXView のファンクションコールでは、レジスタの内容はすべて変更されます。

MSX の BIOS と同様に、将来に渡り MSXView ファンクションコールのエントリは保証されます。

3章 MSXViewの構成と機能

MSXView は次に示すようなマネージャ群により構成されています。

- ディスプレイマネージャ
- ビットブロックマネージャ
- グラフパック
- フォントパック
- テキストマネージャ
- リソースマネージャ
- イベントマネージャ
- コントロールマネージャ
- メニューマネージャ
- ダイアログマネージャ
- プリントマネージャ

この章では、これらのマネージャについて説明します。

3.1 ディスプレイマネージャ

ディスプレイマネージャは、画面表示を管理するマネージャです。MSXView では、オーバーラップウインドウ（相互に重なることのできるウィンドウ）環境を利用することができますが、ディスプレイマネージャは、このウィンドウなどを管理します。MSXView では、画面に対する描画は、ディスプレイマネージャの管理下で行わなければなりません。

一般的に、オーバーラップウィンドウ環境では、他のウィンドウによって隠された別のウィンドウを表に出すためには、アプリケーションがウィンドウを再描画しなければなりません。しかし、MSXView では、ウィンドウの重なりにより失われる部分は、ディスプレイマネー

シャによって自動的に待避されるので、再描画などを行う必要はありません。ただし、重ねられるウィンドウの面積に制限があります。

通常の描画領域にはFWINを、ポップアップメニューなど注目させたい部分にはSWIN BWINを使うことを推奨します。また、通常のウィンドウは、その総面積に限りがあるため、FIXウィンドウという移動やオーバーラップができないウィンドウを使用することができます。各アプリケーションの描画領域、作業領域など大きな面積を使用する部分をウィンドウで管理するには、FIXウィンドウが便利です。FIXウィンドウ同士を重ねあうことはできませんが、FIXウィンドウと通常のウィンドウ(FLOATウィンドウと呼ぶ)を相互に重ねあうことはできます。

3.2 ビットブロックマネージャ

ビットブロックマネージャは、ウィンドウ処理を高速化するために、画面上の矩形領域を効率的に裏VRAMに格納するためのマネージャです。オーバーラップウィンドウの再描画処理をアプリケーションが行わなくても良いのは、ビットブロックマネージャが、隠される部分を自動的に裏画面に待避するからです。ビットブロックマネージャでは、矩形領域を横に1ラインずつスライスし、上から順に格納しています。

通常、ビットブロックマネージャは、ウィンドウ管理の内部処理ルーチンとしてディスプレイマネージャから利用されいますが、画像を扱うアプリケーションなどで直接利用することもできます。

3.3 グラフパック

グラフパックは、グラフィックスの汎用ルーチン群です。描画環境を設定し、点、直線、四角形、円、楕円、扇形、多角形などを高速に描画します。このときに、ペン先や塗りつぶしタイルの指定、クリッピング領域なども指定できます。

グラフパックを使用するには、描画環境(MSXViewではペンと呼ぶ)を設定しなければなりません。

システムでは、SYSPENと呼ばれる標準的な描画環境を用意しているので、通常の描画にはこのSYSPENを使います。しかし、太いペン先や特殊な塗りつぶしタイルなどの特殊な描画環境を使うときは、アプリケーションでペンを作成し、それを使用します。SYSPENを勝手に更新すると、以降のアプリケーションおよびシステムサービスの実行に大きな影響を及ぼしてしまいます。

MSXViewでは、同時に複数のペンを作成しておくことができます。使用中のペンをカレントペンと呼び、他のペンを使用するときには、カレントペンを切り換えます。

グラフパックの使用する座標系は、すべてウィンドウの原点(ウィンドウの左上の点)からの相対座標です。したがって、ウィンドウをどの場所に出しても、内部の描画処理については、気にする必要はありません。

また、グラフパックでは、グラフィックアプリケーションのために、ある点が直線や円の上に存在するかどうかを調べるための機能があります。この機能により、描いた図形をつかむ処理などが簡単に記述できます。

3.4　フォントパック

フォントパックは、文字を出力ためのルーチン群です。複数の種類、大きさのフォントをサポートし、さまざまな飾りつけ（太字、斜体、輪郭、影、縞など）をすることができます。

MSXView では、漢字コードとしてシフト JIS コードを採用しています。これにより、MS-DOS マシンと文書データを交換することができます。

文字の種類は、フォントテンプレートと呼ばれるデータで管理されています。

システムでは、SYSFONT と呼ばれる標準的な文字描画環境を用意しているので、通常の文字表示では SYSFONT を使います。しかし、アプリケーションで特殊な文字を使用する際には、アプリケーションでフォントテンプレートを作成し、それを使うようにして下さい。SYSFONT を勝手に更新すると、以降のアプリケーションおよびシステムサービスの実行に大きな影響を及ぼしてしまいます。

また、MSXView では、同時に複数のフォントを作成しておくことができます。使用中のフォントをカレントフォントと呼びます。他のフォントを使うときは、カレントフォントを切り換えて使用します。

フォントパックの使用する座標系は、グラフパックと同じく、すべてウィンドウの原点（ウィンドウの左上の点）からの相対座標です。したがって、ウィンドウをどの場所に出しても、内部の描画処理については、考慮する必要はありません。

MSXView 漢字 ROM カートリッジは、12×12、12×8 の 2 種類の漢字フォントを内蔵しています（本体に内蔵している機種もある）。その他に、ディスク上にフォントを持つこともできるようになっています。MSXView のシステムディスクには、標準でデザインフォント 4 種類が入っています。フォントパックではこれらのすべてを使うことができます。外字については、GAIJI.MV というファイルに 2 種類のサイズの外字が保存されています。

3.5　テキストマネージャ

テキストマネージャは、文字列の入力と表示を管理します。日本語入力も自動的に処理することが可能です。MSXView で文字列の入力を行うときには、テキストマネージャを使用します。テキストマネージャでは、1 行だけの入力や複数行の入力だけでなく、場合によってはスクロールするような大量の文字列を入力することもできます。

テキストもフォントやペンと同じく複数作成しておくことができ、その内 1 つがカレントテキストと呼ばれます。キーボードは 1 つしかないので、キーからの入力は必ずカレントテキストに対しての入力とみなして処理されます。

文字列編集は、原則としてキーボードを使用して行うようになっていますが、マウスなどのポインティングデバイスを使うことも考慮されており、コントロールマネージャと併用し

て、マウスによるカーソル位置の変更や、領域指定が簡単に行えます。

また、いくつかの文字列から1つを選ぶというアイテムセレクタについても、テキストマネージャを使用することにより、非常に簡単に作成できます。例えば、VSHELLではファイル選択、プリンタ選択などで、テキストマネージャによるアイテムセレクタを使っています。

3.6 リソースマネージャ

リソースマネージャは、ファイルの管理を行うマネージャです。MSX-DOS環境では手間のかかるエラー処理などもサポートしています。

また、通常のMSX DOSファイルシステムの呼び出し機能の他に、最大64Kバイトのファイルバッファを操作できるファイルアロケータを管理する機能も備えています。MSXViewでは、アプリケーションエリアが最大32Kバイトなので、大きなデータは、ファイルアロケータを使用して、ディスク上に取らなければなりません。

また、オーバーレイプログラムの実行をサポートするための機能も、リソースマネージャの管理下にあります。

3.7 イベントマネージャ

イベントマネージャは、システムに対する外部からの入力（キーボード、ポインティングデバイスなど）を管理し、これらをイベントとして扱うマネージャです。MSXViewでは、アプリケーションの動作は、外部からのイベント（キーボードからの入力やポインティングデバイスのボタン操作など）によって起動されます。

また、イベントマネージャは、割り込み処理を使って、ポインティングデバイスの読み込みを行い、これにより移動するマウスカーソルの表示処理を行います。したがって、アプリケーションではマウスカーソルの管理、表示について気にする必要はありません。

マウスカーソルは、最大16×16ドットの任意のパターンが使用できます。色は2色使えます。マウスカーソルの形状は最大12個まで登録でき、そのうち2つは、システムで予約されています。

マウスカーソル番号3番以降は、アプリケーションが自由な形状を割り当てて使うことができます。また、領域を指定しておけば、指定された領域に入ったときだけカーソルのパターンを変える機能もあり、最大16個の領域を指定しておくことができます。マウスカーソルは割り込み処理で自動的に描かれ、指定された領域に入ったら、自動的に指定された形状に変化します。

イベントは、イベントレコードという形で、システムが管理するイベントキューにためられるようになっています。アプリケーションは、必要に応じてイベントマネージャを呼び出し、イベントレコードをもらい、そのイベントの種類別に分岐して、イベントの処理を行うのが一般的です。イベントレコードには、そのイベントが生じたときの SHIFT キーなどの状態やマウスカーソルの位置が保存されているので、必要な情報は、ほとんどの場合、イベントレコードから得ることができます。

また、イベントマネージャはイベント待ちの間に、矩形領域を点滅させること（ウィンカと呼ぶ）ができます。ウィンカは、一般的には文字列編集時のテキストカーソルやアイテムセレクタのカーソルに使用します。

ウィンカはイベント待ちの間点滅し、イベントが生じると自動的に消えるので、アプリケーションはウィンカのオン・オフを気にする必要はありません。ウィンカは、システムで同時に複数個管理できますが、フォントやペン、テキストと同様に1つだけカレントウィンカが存在し、同時に点滅するウィンカは1つだけです。

テキストマネージャを使用する場合には、テキストカーソルとしてウィンカを自動的に設定するので、アプリケーションは何もする必要はありません。

イベントマネージャでは、マウスを持っていない人のために、[GRAPH]+[SELECT]キー、[STOP]キー、カーソルキー（[↑][↓][←][→]）を押すことにより、マウスの1st ボタンと2nd ボタン、移動をシミュレートできるようになっています。

3.8 コントロールマネージャ

コントロールマネージャは、画面に表示されるさまざまなコントロール（ツマミ類）を管理するマネージャです。コントロールとは、例えば画面に表示されるボタン、チェックマーク、スクロールバーなどのもので、このマネージャを使用することにより、これらの管理を容易にかつ効率的に行うことができます。

コントロールを使用するには、コントロールテンプレートというデータ構造に基づいて、データを作成しておかなければなりません。コントロールテンプレートには、コントロールの種類と状態、位置（ローカル座標）、大きさなどが格納されます。このコントロールテンプレートを用意しておくと、コントロールの表示やマウスカーソルの位置を指定したコントロールの検索、実際のコントロールの動作処理などが簡単に記述できます。

また、アプリケーションで特殊なコントロール（ツマミ）を作成し、登録することもできます。MSXView では、これをカスタムコントロールと呼んでいます。この場合は、コントロールドライバというモジュールを MSXView のルールにしたがって作成し、そのエントリを登録しなければなりません。

3.9 メニューマネージャ

メニューマネージャは、文字列メニューを管理するマネージャです。通常のアプリケーション開発では、ユーザーが扱いやすいメニューの管理はたいへん面倒です。MSXView では、メニュー内に表示する文字列を所定のデータ形式で並べるだけで、メニューの大きさや配置にいたるまで、メニューマネージャが管理します。メニューはポインティングデバイスを使用してもキーボードを使用しても簡単に選択できるようになっています。

メニューマネージャでは、プルダウンメニュー、ポップアップメニューなどの各種メニューを管理することができます。

原則としてメニューは、現在のカーソル位置を中心とした位置に表示され、横は画面いっぱいに、縦はメニューバーに重ならないよう、18ドットから211ドットまでに自動的に制限されます。固定位置に出現するようにすることもできます。ポップアップメニューの大きさは、メニューテンプレートの内容により、自動的に最小の大きさに設定されます。

メニューの表示内容は、基本的には文字列で構成されます。また、メニューをキーボードを使ってワンタッチで選択できるようにするため、メニューの要素ごとにキーコードが指定できます。

3.10 ダイアログマネージャ

ダイアログマネージャは、ユーザーとの対話を管理するマネージャです。一定のデータを用意しておくだけで、ユーザーからのイベントを自動的に受けつけ、処理を行います。ダイアログマネージャを使うことにより、コントロールマネージャの扱いがより簡単になります。

3.11 プリントマネージャ

MSXView では、サポートしているプリンタごとに、プリンタドライバという特殊なオーバーレイモジュールが用意されています。これにより、すべてのプリンタを「MSXView 仮想プリンタ」として統一的に扱うことができ、プリンタの種類ごとに別々のプログラムを開発していた従来の方法に比べて、大変簡単に各種のプリンタをサポートすることができます。

基本的には、VRAM にイメージ情報を展開し、これをプリンタドライバに渡して、実際に印字を行うという手順になります。

3.12 その他のマネージャ

MSXView には、以上で簡単に説明したマネージャの他にも、以下のようなマネージャがあります。

- システムマネージャ
- キーマップマネージャ
- サウンドマネージャ
- メモリマネージャ
- プリントマネージャ

4章
ハンドルの概念

MSXView では、さまざまなマネージャでのデータブロックの指定に、ハンドルと呼ばれるデータ構造を使用しています。ハンドルは1バイトの整数で表現され、その番号により、以下の意味があります。

表 3.1 ハンドルの意味

ハンドル番号	意味
0	ハンドルの番号としては使用しません。 システムに対して、新しいハンドルを割り付けを要求するときなどに使用します。
1～127	指定した番号を割り当てて、固定的に使用します。 システムがアプリケーション用に固定的に提供しているものや、アプリケーションのメインモジュールとオーバーレイモジュールとのリンクに使用します。ただし、汎用のモジュールではハンドルの割り当てが重なる恐れがあるため、この固定ハンドルを用いることはできません。
128～254	他のハンドルと重ならないように、動的に割り当てられる番号です。 オーバーレイモジュール、デスクアクセサリ、汎用モジュール内などで一時的に使われたり、アプリケーションが通常使用するハンドルてす。
255	ハンドルの割り当てに失敗した場合のエラーとして返す番号です。 ハンドルとしては使用しません。

MSXView では、次のようなものがハンドルで管理されています。

- ウインドウ
- ビットブロック
- ペン（グラフパックの描画環境）
- フォントテンプレート（フォントパックの描画環境）

- テキストテンプレート（文字列編集の環境）
- ファイル
- メニュー
- コントロール
- ウィンカ

MSXViewでは、ハンドル管理されているデータブロックは、すべてシステムエリアに規定個数のデータエリアが用意されています。したがって、アプリケーションが各種のテンプレートとして、大きなデータブロックをいくつも準備する必要はありません。

ハンドル管理を行うことにより、大量のデータを1バイトで指定できるので、貴重なアプリケーションプログラム領域を無駄使いすることなく、プログラムのオーバーレイなどを行っても、データの位置を気にすることなく開発することができます。

5章 APの標準レイアウト

アプリケーションの画面レイアウトは、MSXView システムの全体的な統一化のために特に不都合のないかぎり以下の形式にして下さい。

これにより、ユーザーはある1つのアプリケーションの基本操作を覚えてしまえば、他の新しいアプリケーションも無理なく使えるようになるというメリットが生まれます。

5.1 タイトルバー

ファイル処理、印刷、終了などのメニューが入っています。また、現在編集中のファイル名の表示も行います。編集中のファイルが新規作成中の場合には、「新規」と表示されます。

ここに入るメニューの内容は以下のようなものです。

表 3.2 タイトルバーの内容

内容	意味
新規	編集内容を破棄して初期状態にします。
保存	編集内容に名前をつけて保存します。
更新	編集内容を読み込んだときの名前で保存します。
読込	保存してあったデータをディスクから読み込みます。
登録	標準形式のデータ交換ファイルを作ります。
組込	標準形式のデータ交換ファイルを組み込みます。
印刷	印刷を行います。
印刷形式	印刷形式を設定します。
終了	アプリケーションを終了し、VSHELL に戻ります。

5.2 DA バー

デスクアクセサリメニューが入っています。

5.3 コマンドバー

アプリケーションの各メニューが入っています。この部分のメニューはアプリケーションによって異なります。

図 3.1 メニューバーの各部の名称

6章
操作における規定事項

この章では、MSXView 用のアプリケーションを開発するにあたって、操作面で守らなければならないことを説明します。

6.1　操作方法

文字の入力以外はマウスのみで操作可能にして下さい。また、すべての操作をキーボード上からも行えるように配慮して下さい。

6.2　デスクアクセサリ

ほとんどの状態でデスクアクセサリが起動できなくてはいけません。

6.3　特殊キー

ファンクションキーやグラフキーのような特殊キーは、以下のように割り付けます。

[F1]～[F10]	コマンドバーの左から順に割り付けます。
[GRAPH]＋[文字]	頻繁に使う機能を割り付けます。

6.4　印刷

印刷については、MSXView で定められている仮想プリンタを対象にプログラムを作成します。1つ1つのプリンタの差は、MSXView で用意されている各種のプリンタドライバによって吸収されるので、多数のプリンタを対象にプログラムを開発する必要はありません。

6.5　ファイル名の入力

読み込み、保存時のファイル名の入力はシステムで用意されている FILEPACK を使用して下さい。これによりユーザーは、すべてのアプリケーションプログラムで共通して、読み込み、保存を同じ操作方法で扱うことができます。

FILEPACK の具体的な使用法は、添付のサンプルプログラムを参照して下さい。

7章
ファイルの形式

この章では、MSXViewのファイル形式について説明します。

7.1 アプリケーションファイル

アプリケーションファイルは、先頭からアプリケーションエリアに読み込まれるバイナリファイルでなければなりません。さらに、オーバーレイモジュールを含むアプリケーションの場合には、ファイルの後ろにオーバーレイモジュールが連結されていなければなりません。

7.2 データファイル

アプリケーションが作成するデータファイルの形式は自由です。したがって、どのようなフォーマットでファイルを作成してもかまいません。また、他のアプリケーションやMSXView以外の各種アプリケーションとデータの互換性を持たせるために、複数のファイルフォーマットをサポートすることもできます。この場合には、タイトルバーの中に、ファイル形式変更のメニューがなければなりません。

また、データファイルの中にはユーザーが作成したデータの他に、そのデータが保存されたときのアプリケーションの状態を示すデータを入れておくと便利です。例えば、表示形式や表示位置、パレット、タイル、ペンの形などが保存されていれば、そのファイルを読み込むだけで、保存したときと同じ作業状態になるので、ユーザーがすぐに仕事にとりかかれます。

ただし、MSXViewでは、複数アプリケーション間でのデータの互換性を保証するため、「登録」、「組込」コマンドで、MSXView標準データフォーマットに基づくファイルが作成できなければなりません。MSXView標準データフォーマットについては、10章「MSXView標準データ」を参照して下さい。

8章
オーバーレイプログラムの作成

MSXViewシステムでは、アプリケーションエリアが32Kバイトなので、ほとんどのアプリケーションプログラムは、オーバーレイプログラムを用いることになります。MSXViewでは、オーバーレイプログラムを別のファイルにはしないで、アプリケーションの実行形式ファイルの後ろに、オーバーレイモジュールを付加するというファイル構造をとっています。したがって、アプリケーションを1つのファイルとして扱えるようになっています。このため、オーバーレイモジュールを作成した後、アプリケーションプログラムとオーバーレイモジュールを1つのファイルにまとめなければなりません。

また、アプリケーションを設計するときに、どの部分をオーバーレイモジュールにするかで、開発効率やメンテナンスの手間が大幅に変わってきます。設計時には十分に検討して下さい。

オーバーレイモジュールは以下の事項にそって作成します。

1. 独立性
2. 単機能

8.1 独立性

オーバーレイで非常に困難なのが常駐部とのリンクです。なるべくなら、リンクをしなくても動くようなオーバーレイモジュールを作成することがよいでしょう。その方が保守性がよく、いろいろなアプリケーションで共通に使うことができます。

ただし、ユーザーインターフェイスの部分は常駐部に入れておかないと、ユーザーへのリアクションが素早く行えません。この部分は、できるだけオーバーレイにはしないで下さい。例えば、ボタンを押して強調印字するのは常駐部で、その後の実際の機能（処理）はオーバーレイにします。ユーザーがボタンを押したら、それに対するリアクションがすぐに返らなくては、使い心地の良いアプリケーションにはなりません。

8.2　単機能

オーバーレイモジュールは、できるだけ機能ごとに小さく分割することを推奨します。

1つのオーバーレイモジュールにたくさんの機能を持たせ大きくすると、ユーザーが1つの機能しか利用しないときでも、バッファが大量に取られ、メモリの使用効率が低下するからです。

しかし、ユーザーの必要としない機能までバッファに入ることが問題なのですから、そのときに絶対に必要とする機能ならば複数の機能を入れた方が良いでしょう。

9章 MSXView基本データ構造

この章では、MSXView基本データ構造について説明します。

9.1 1バイト型の別名定義と定数名

1バイト型の別名定義と定数名は、次のように定義されています。

```
#define TINY    char          /* 0~255 を表す数値型 */

#define BOOL    char          /* 論理型 */
#define TRUE    1             /* (~FALSE) ではなく (!FALSE) */
#define FALSE   0             /* C 言語の論理式の値に対応 */

#define STATUS  char          /* 成功・失敗型 */
#define OK      0
#define ERROR   0xff

#define HANDLE  char          /* ハンドル型 */
#define NEW     (char)0       /* 新規ハンドルの割り当て要求 */
#define ROOTBD  1             /* ルートボード */
#define SYSPEN  1             /* 標準ペン */
#define BDPEN   2             /* ルートボードのペン */
#define SYSFONT 1             /* 標準フォント */

#define COLOR   char
```

9.2 基本的な構造体

画面表示で使用する構造体は、以下のように宣言されています。

```
typedef struct          _pos {  /* 座標の指定に使用（主に画面座標） */
        int             xp;     /*   負の座標はウィンドウ外なので、 */
        int             yp;     /*   表示されないが論理的には有効 */
```

```
}       POS;

typedef struct          _area { /* POS+領域サイズ */
        int             xp;     /*   本質的には POS だが、この方が */
        int             yp;     /*   初期化をシンプルに書ける。 */
        unsigned        xs;     /*   サイズは非負なので論理的だが */
        unsigned        ys;     /*   計算時の型変換には要注意。 */
}       AREA;
```

9.3 ディスプレイマネージャの構造体

ディスプレイマネージャで使用する構造体は、以下のように宣言されています。

```
typedef struct _win {           /* ウィンドウの状態 */
        HANDLE  block;          /* ビットブロック (FIXなら0) */
        TINY    status;         /* ウィンドウスタイル+ウィンドウの種類 */
        AREA    area;           /* グローバル座標 */
        HANDLE  defpen;         /* デフォルトペン */
        HANDLE  deffont;        /* デフォルトフォント */
        HANDLE  curpen;         /* カレントペン */
        HANDLE  curfont;        /* カレントフォント */
}       WIN;
```

9.3.1 ウィンドウスタイルの定数

ウィンドウの形状を指定する定数は、次のように定義されています。

```
#define FIX     (char)128       /* ビットブロックな */
#define TWIN    (char)0         /* なにもなし */
#define CWIN    (char)1         /* 枠なし */
#define FWIN    (char)2         /* 一重枠 */
#define RWIN    (char)3         /* 角の丸い一重枠 */
#define IWIN    (char)4         /* インデックス枠 */
#define SWIN    (char)5         /* 影付き一重枠 */
#define BWIN    (char)6         /* 影付き二重枠 */
```

9.4 ビットブロックマネージャの構造体

ビットブロックマージャで使用する構造体は、以下のように宣言されています。

```
typedef struct _blcinfo {
        TINY    lotnum;         /* ロット番号 */
        LONG    start;          /* VRAMポインタ */
        LONG    length;         /* ブロックデータの長さ */
        WORD    xs;             /* 横方向の大きさ */
```

```
        WORD    ys;                 /* 縦方向の大きさ */
}       BLCINFO;
```

9.5 イベントマネージャの構造体

イベントマージャで使用する構造体は、以下のように宣言されています。

```
typedef struct  _event {            /* イベントレコード */
        TINY    kind;               /* イベントの種類 */
        POS     where;              /* カーソルのグローバル座標 */
        TINY    bstat;              /* ボタン（含むシフトキー）の状態 */
        char    keycode;            /* シフトを無視したコード */
        TINY    kstat;              /* シフト（含む特殊キー）の状態 */
        TINY    keymap;             /* キーマップ・コード */
        TINY    msg[4];             /* 予備 */
}       EVENT;
```

9.5.1 イベントの定数

イベントの種類を指定する定数は、次のように定義されています。

```
#define KEYEVT          1               /* キーが押された */
#define TRIGDN          2               /* 1st ボタンが押された */
#define TRIGUP          3               /* 1st ボタンが離された */
#define ABORT           4               /* 2nd ボタンが押された */
#define ABORTUP         5               /* 2nd ボタンが離された */
#define KANJI           6               /* 予約 */
```

9.5.2 ウィンカの構造体

ウィンカの状態を表す構造体は、次のように宣言されています。

```
typedef struct          _wink {         /* ウィンカ */
        HANDLE          win;            /* 所属するウィンドウ */
        AREA            area;           /* ローカル座標 */
        unsigned        speed;          /* 点滅スピード（ビット指定） */
        TINY            rev;            /* 反転時の xor 値 (0xfe) */
}       WINK;
```

9.5.3 キーボードの構造体

キーボードの状態を表す構造体は、次のように宣言されています。

```
typedef struct       _locks {        /* キー[illegible]ドの[illegible] */
        TINY         config;         /* キー配列 (0~[illegible]  */
        TINY         kana;           /* かなロック */
        TINY         caps;           /* CAPS ロック */
}       LOCKS
```

9.5.4 マウスカーソルの構造体

マウスカーソルの状態を表す構造体は、次のように宣言されています。

```
typedef struct       _cursor {       /* マウスカーソルの形状 */
        TINY         xhot;           /* pat 内のどのドットを中心座標 */
        TINY         yhot;           /* とするかを指定 (0~7)  */
        TINY         logic;          /* 画面との論理演算 */
        COLOR        col1;           /* pat1 の色 */
        TINY         pat1[32];       /* 16 × 16 のビットパターン */
        COLOR        col2;           /* pat2 の色 */
        TINY         pat2[32];       /* 16 × 16 のビットパターン */
}       CURSOR;
```

9.6 グラフパックの構造体

グラフパックで使用する構造体は、次のように宣言されています。

```
typedef struct  _rgb {          /* パレットの色指定などで使用する */
        TINY    red;            /* 0~7 */
        TINY    green;          /* 0~7 */
        TINY    blue;           /* 0~7 */
}       RGB;

typedef struct  _tile {         /* PEN 構造体のためのサブ構造体 */
        TINY    sw;             /* 色づけに関する指定 */
        COLOR   on;             /* pat の on ドットの色 */
        COLOR   off;            /* pat の off ドットの色 */
        TINY    pat[8];         /* 8 × 8 のビットパターン */
}       TILE;                   /* TILE.sw の意味は次のとおり */

typedef struct  _grafpen {      /* 文字以外の描画の基本となる */
        TINY    line[4];        /* ラインスタイル */
        TINY    xhot;           /* pat 内のどのドットを中心座標 */
        TINY    yhot;           /* とするかを指定 (0~7)  */
        TINY    xs;             /* pat(ペン先)のサイズを指定 */
        TINY    ys;             /* (0~7) */
        TINY    pat[8];         /* ペン先のパターン */
        TILE    pen;            /* 点、線の描画用 */
        TILE    fill;           /* 塗りつぶし用 */
```

```
        TILE    back;           /* ウィンドウ消去用 */
}       PEN;
```

TILE.sw に指定する定数

TILE.sw で使用する定数は、次のように定義されています。

```
#define ONDOT   (char)32        /* onドッ[illegible]功 */
#define OFFOOT  (char)16        /* off ドット有功 */
                                /* 有効でないドットは透明扱い */
                                /* (ONDOT|OFFOOT) は2色タイル */
                                /* sw=0 なら on 色のベタ塗り */
```

9.7 フォントパックの構造体

フォントパックの構造体は、次のように宣言されています。

```
typedef struct  _font {         /* フォントテンプレート */
        char    id;             /* フォントID */
        TINY    width;          /* サイズ 横ドット数 */
        TINY    hight;          /* サイズ 縦ドット数 */
        TINY    boldx;          /* 太字 横ドット数 */
        TINY    boldy;          /* 太字 縦ドット数 */
        TINY    italic;         /* 斜体 (0~32) */
        TINY    shadow;         /* 影の数 (0~4) */
        COLOR   shadowc[4];     /* 影の色 */
        TINY    outline;        /* 輪郭の数 (0~4) */
        COLOR   outlinec[4];    /* 輪郭の色 */
        COLOR   underline;      /* 下線の色 (0は下線なし) */
        BOOL    stripe;         /* 縞 */
        COLOR   fcol;           /* 文字色 */
        COLOR   bcol;           /* 未使用 */
        TINY    pitch;          /* 文字間 */
        TINY    logic;          /* 未使用 */
        TINY    direc;          /* ビット7~ビット4  未使用 */
}       FONT;                   /* ビット3  シフトJIS禁止 */
                                /* ビット2  プロポーショナル禁止 */
                                /* ビット1  文字の回転 */
                                /* ビット0     〃      */

typedef struct  _fntmsg {       /* 直前の文字の情報 */
        TINY    s_base;         /* 元文字のベースライン */
        TINY    s_width;        /* 元文字の width */
        TINY    s_hight;        /* 元文字の hight */
        TINY    t_base;         /* 実際のベースライン */
        TINY    t_width;        /* 実際の width */
        TINY    t_hight;        /* 実際の hight */
```

```
        TINY    d_base;         /* 飾りつけ後のベースライン */
        TINY    d_width;        /* 飾りつけ後の width */
        TINY    d_hight;        /* 飾りつけ後の hight */
        TINY    f_pitch;        /* 左にはみだすドット数 */
        TINY    b_pitch;        /* 右にはみだすドット数 */
}       FNTMSG;
```

9.8 テキストマネージャの構造体

テキストマネージャの構造体は、次のように宣言されています。

```
typedef struct          _text {         /* テキストテンプレート */
        HANDLE          win;            /* 所属するウィンドウ */
        AREA            area;           /* ローカル座標 */
        char            *buff;          /* テキストバッファ */
        int             length;         /* テキストバッファの長さ */
        WORD            opt;            /* 各種オプション */
        HANDLE          font;           /* フォント */
        unsigned        line;           /* 行数(0は無制限) */
        char            *end;           /* バッファ終端 */
        unsigned        lines;          /* バッファ内の行数 */
        unsigned        topline;        /* 表示中の先頭行 */
        char            *cursor;        /* カーソル位置 */
        unsigned        column;         /* カーソルの論理X座標 */
        unsigned        row;            /* カーソルの論理Y座標 */
        char            *range;         /* 選択範囲の先頭 */
        char            *endrange;      /* 選択範囲の終端 */
}       TEXT;
```

9.9 メニューマネージャの構造体

メニューマネージャの構造体は、次のように宣言されています。

```
typedef struct  _popup {        /* ポップアップテンプレート */
        TINY    head;           /* 各種スイッチ */
        char    keycode;        /* ショートカットキーの指定 */
        char    *name;          /* 項目名文字列 */
}       POPUP;

typedef struct  _menutp {       /* メニューテンプレート */
        TINY    head;           /* 各種スイッチ */
        char    keycode;        /* ショートカットキーの指定 */
        char    *name;          /* 項目名文字列 */
        POPUP   *popup;         /* 選択時に実行するポップアップ */
}       MENU;
```

```
typedef struct  _command {       /* メニュー、ダイアログなどの選択結果 */
        TINY    menu;            /* メニューハンドル */
        TINY    item;            /* これらの項目は関数によって */
        TINY    func;            /* 意味が異なるので注意が必要 */
}       COMMAND;
```

9.9.1 POPUP head および MENU head に設定する定数

POPUP.head および MENU.head に設定する定数は、次のように定義されています。

```
#define CHK     (char)128       /* チェックマーク */
#define MSK     (char)64        /* 表示、選択禁止 */
#define DIS     (char)32        /* 選択禁止 */
#define LIN     (char)16        /* 線（選択禁止） */
#define CEN     (char)8         /* センタリング */
#define BLD     (char)4         /* 太字 */
#define CNT     (char)2         /* 改行せずに継続する */
#define FIN     (char)1         /* 終了（最後の要素である印） */
#define NON     (char)0         /* 何も指定しない場合 */
```

9.10 コントロールマネージャの構造体

コントロールマネージャの構造体は、次のように宣言されています。

```
typedef struct          _ctrltp {       /* コントロールテンプレート */
        TINY            number;         /* コントロール番号 */
        TINY            sw;             /* MENU.head と同様の意味。 */
        int             xp;             /* 本質的には AREA であるが、 */
        int             yp;             /* スタティックに初期化するのが */
        unsigned        xs;             /* 通常の使い方なので、シンプル */
        unsigned        ys;             /* に書けるようにわけてある。 */
        char            *msg;           /* 文字列コントロールメッセージ */
}       CONTROL;
```

9.10.1 標準コントロール番号の定数

標準コントロール番号は、次のように定義されています。

```
#define NULL_CNTL       1       /* NULL */
#define BUTTON_CNTL     2       /* ボタン */
#define MARK_CNTL       3       /* チェックマーク */
#define CNTL_CNTL       4       /* コントロールボタン */
#define HBAR_CNTL       5       /* 横スクロールバー */
#define VBAR_CNTL       6       /* 縦スクロールバー */
#define CICON_CNTL      7       /* 色付きアイコン */
#define ICON_CNTL       8       /* アイコン */
```

```
#define FRAME_CNTL      9       /* フレーム */
#define LINE_CNTL       10      /* 線 */
#define ROUND_CNTL      11      /* 角の丸い四角 */
#define ERASE_CNTL      12      /* 1 色で塗りつぶす */
#define STRING_CNTL     13      /* 文字列 */
#define TEXT_CNTL       15      /* テキ[illegible] */

/* カレント pen、font を使用するコントロール */
#define BUTTON_STD      17
#define MARK_STD        18
#define CNTL_STD        19
#define HBAR_STD        20
#define VBAR_STD        21
#define ICON_STD        22
#define FRAME_STD       23
#define LINE_STD        24
#define ROUND_STD       25
#define STRING_STD      26

typedef struct  _bartmp {       /* スクロールバーテンプレート */
        int     curnum;         /* 現在の位置 (0 origin)  */
        int     maxnum;         /* 全体の行数 (1 origin)  */
        int     pagenum;        /* 1 ページの行数 (1 origin)  */
}       BARTMP;

typedef struct  _cicon {        /* 2色アイコンテンプレート */
        COLOR   on;             /* on ドットの色 */
        COLOR   off;            /* off ドットの色 */
        TINY    *pat;           /* ビットパターン */
}       CICON;
```

9.10.2 コントロールのパート番号の定数

コントロールのパート番号は、次のように定義されています。

```
#define ALL     (char)0         /* 全体を示す */
#define PAGEL   1               /* HBAR_CNTL  左ページ */
#define PAGER   2               /*            右ページ */
#define MOVEH   128             /*            レバー   */
#define PAGEU   3               /* VBAR_CNTL  前ページ */
#define PAGED   4               /*            次ページ */
#define MOVEV   129             /*            レバー   */

typedef union           _msg {  /* コントロールメッセージへのポインタ */
        int             i;      /*  これらは CONTROL.msg に入る、 */
        int             *ip;    /*  すべてのポインタの unionだが、 */
        unsigned        u;      /*  CONTROL にこれを直接入れると */
        unsigned        *up;    /*  スタティックな初期化ができな */
```

```
        char          c;      /*  いので、後で一般形の CNTL を */
        char          *p;     /*  定義している */
        HANDLE        h;
        HANDLE        *hp;
        TINY          t;
        TINY          *tp;
        POS           *pos;
        AREA          *area;
        BARTMP        *bar;
        PEN           *pen;
        FONT          *font;
        RGB           *rgb;
        TILE          *tile;
        CONTROL       *cntl;
        TEXT          *text;
}       MSG;

typedef struct  _cntl {         /* コントロールテンプレートの一般形 */
        HNDLE   number;         /*  これはスタティックな初期化が */
        TINY    sw;             /*  できないが、式の中では後述の */
        int     xp;             /*  キャスト用マクロで使用する。 */
        int     yp;             /*  それ以外に CNTL 型を使用する */
        WORD    xs;             /*  ことはほとんどない。 */
        WORD    ys;
        MSG     msg;            /* 一般のコントロールメッセージ */
}       CNTL;
```

9.10.3 コントロールメッセージへのキャスト用マクロ

コントロールメッセージへのキャスト用マクロは、次のように定義されています。

```
#define _(C)    (*(CNTL *)&C)   /* _(CONTROL 配列要素).msg.foo */
#define __(C)   ((CNTL *)C)     /* __(CONTROL ポインタ)->msg.foo */
```

9.10.4 コントロールドライバのバリエーション番号

コントロールドライバのバリエーション番号は、次のように定義されています。

```
#define DRAW_CD         0
#define SEL_CD          1
#define FIND_CD         2
#define CATCH_CD        3
#define DLVR_CD         4
#define ELVR_CD         5
#define FREE_CD         6
#define EXIT_CD         7
#define CHKEY_CD        8
#define KEY_CD          9
```

```
#define OPEN_CD         10
#define CLOSE_CD        11
```

9.11 プリンタドライバの構造体

プリンタドライバの構造体は、次のように宣言されています。

```
typedef struct      _print {        /* プリントテンプレート */
        char        *buff;          /* プリントバッファへのポインタ */
        char        prname[16];     /* プリンタ名 */
        TINY        id;             /* プリンタ ID */
        int         inch;           /* 1インチのドット数 */
        int         line;           /* 印字1行のドット数 */
        char        papname[16];    /* ペーパーの名前 (A4、A5など) */
        TINY        pid;            /* ペーパー ID */
        int         width;          /* 1ページの横ドット数 */
        int         hight;          /* 1ページの縦ドット数 */
        int         column;         /* 1ページの桁数 */
        int         row;            /* 1ページの行数 */
        unsigned    opt;            /* 縦・横、トラクタ・カットシート */
        int         startp;         /* 開始ページ */
        int         endp;           /* 終了ページ */
        int         copy;           /* 印刷枚数 */
        TINY        msg[32],        /* メッセージ */
}       PRINT;
```

9.11.1 プリンタドライバの機能コード

プリンタドライバの機能コードは、次のように定義されています。

```
#define PD_INIT     0       /* プリンタドライバの初期化 */
#define PD_OPEN     1       /* プリンタの印刷開始宣言 */
#define PD_CLOSE    2       /* プリンタの印刷終了宣言 */
#define PD_PRINT    3       /* 印刷の実行 */
#define PD_MENU     4       /* 用紙の設定 */
#define PD_START    5       /* 開始・終了ページ、印刷枚数の設定 */
#define PD_PAGE     6       /* 印刷開始メッセージの表示 */
```

9.12 その他の構造体

その他の構造体は 次のように宣言されています。

9.12.1 オーバーレイで使用する構造体

オーバーレイで使用する構造体は、次のように宣言されています。

```
typedef struct _module {        /* オーバーレイなどで使用する */
        long    p;
        long    size;
}       MODULE;
```

9.12.2 モジュール名の宣言

モジュール名は次のように定義されています。

```
#define   _public(N)    static char       ___/**/N[ ] = "**_pub_**:N"; ¥
                        static MODULE     *N = (MODULE *)___/**/N
#define   _extern(N)    static char       ___/**/N[ ] = "**_ext_**:N"; ¥
                        static MODULE     *N = (MODULE *)___/**/N
```

9.12.3 日付を表す構造体

日付を表す構造体は、次のように宣言されています。

```
typedef struct  _date {
        int     year;           /* 1980～2079 */
        TINY    month;          /* 1～12 */
        TINY    date;           /* 1～31 */
        TINY    week;           /* 0=日曜、… 6=土曜 */
}       DATE;
```

9.12.4 時間を表す構造体

時間を表す構造体は、次のように宣言されています。

```
typedef struct  _time {
        TINY    hour;           /* 0～23 */
        TINY    minute;         /* 0～59 */
        TINY    second;         /* 0～59 */
        TINY    sec100;         /* システム予約 */
}       TIME;
```

9.12.5 MSX–DOS の DPB を表す構造体

MSX–DOS の DPB を表す構造体は、次のように宣言されています。

```
typedef struct          _dpb {          /* MSX-DOS の DPB */
        TINY            drive;          /* Drive number */
        TINY            media;          /* Media ID */
        unsigned        sectsize;       /* Size of 1 sector */
        TINY            dirmask;        /* directory mask */
```

```
        TINY            dirsft;         /* directory shift */
        TINY            clusmask;       /* Cluster mask */
        TINY            clussft;        /* Cluster shift */
        unsigned        topfat;         /* Top sector of FAT */
        TINY            fatnum;         /* Number of FAT */
        TINY            direntry;       /* Directory entry */
        unsigned        topdata;        /* Top sector of data area */
        unsigned        cluster;        /* Number of cluster +1 */
        TINY            sectclus;       /* Sector par cluster */
        unsigned        topdir;         /* Top sector of directory entry */
        TINY            fat;            /* Top address of FAT */
        unsigned        free;           /* Free cluster */
}       DPB;
```

9.12.6 MSX–DOS の FCB を表す構造体

MSX-DOS の FCB を表す構造体は、次のように宣言されています。

```
typedef struct          _fcb {          /* MSX-DOS の FCB */
        TINY            drive;          /* Drive name. 0=default, 1=A */
        TINY            name[8];        /* Filename */
        TINY            ext[3];         /* Extention */
        unsigned        crblk;          /* Current block */
        unsigned        recsiz;         /* Record size */
        long            size;           /* File size */
        unsigned        date;           /* Created date */
        unsigned        time;           /* Created time */
        TINY            devid;          /* device ID */
        TINY            dirloc;         /* Directory location */
        unsigned        first;          /* First cluster of a file */
        unsigned        last;           /* Last cluster of a file */
        unsigned        access;         /* Last cluster accessed */
        TINY            crrec;          /* Current record */
        long            random;         /* Random record */
}       FCB;
```

10章 MSXView標準データ

この章では、MSXViewの標準データについて説明します。

10.1 MSXView標準データとは

MSXViewでは、あるアプリケーションで作成したデータを、他のアプリケーションでも利用できるように、標準データファイルのフォーマットが規定されています。したがって、標準データの登録・組込の機能をサポートするMSXViewアプリケーションの間では、データを自由にやりとりすることができます。

標準データファイルはヘッダ部、プライベートデータ部、スタンダードデータ部の3つの部分から構成されています。

ヘッダ部は、その標準データを作成したアプリケーションのIDとプライベートデータ部の長さを記録します。

プライベートデータ部は、その標準データを作成したアプリケーションが自由に使うことができます。これによって、同一アプリケーションが作成した標準データは完全に元の状態のまま再利用することができます。

スタンダードデータ部は、MSXViewで一般的に使用される描画ファンクションのシーケンスとして、データを記録します。そのため、アプリケーションごとに使用するスクリーンモードが違っていたり、ViewDRAWとViewPAINTのようにデータの表現形式が異なっていても、元のイメージをできるかぎり忠実に再現することができます。

標準データを解釈して実際の描画を行うのは、アプリケーションの責任です。データを組み込もうとしたアプリケーションが、標準データのすべてを取り扱えないときは、元のイメージを再現することができない場合もあります。例えば、ViewCALCで作成した円グラフをViewDRAWに組み込もうとしたとき、ViewDRAWには円弧を取り扱う機能がないために円グラフを表示することはできません。

10.2 標準データファイルのフォーマット

ここでは、標準データファイルを構成するヘッダ、プライベートデータ、スタンダードデータのそれぞれの内容を解説します。

10.2.1 ヘッダ

ヘッダ部の構成を説明します。

```
0000:    char    id[2]          ; アプリケーションID
0002:    WORD    length         ; プライベートデータ長
```

id には標準データファイルを作成したアプリケーションの ID を、length にはプライベートデータのサイズを 16 ビットの数値で記録します。

この ID は登録制で、株式会社アスキーが管理します。したがって、アプリケーションソフトウェアを販売・頒布するときは、あらかじめ弊社に連絡し、ID の割り当てを受けて下さい。

アプリケーションごとの ID の例を以下に示します。

表 3.3 標準データの ID

アプリケーション	ID
ViewDRAW	DR
PageEDIT	DR
ViewPaint	BT
ViewTED	TD
ViewCALC (グラフ)	CH
ViewCLAC (シート)	CL

10.2.2 プライベートデータ

プライベートデータはヘッダの直後に記録します。フォーマットは、各アプリケーションごとに異なります。

10.2.3 スタンダードデータ

スタンダードデータは、プライベートデータの直後に記録します。その位置は、length にヘッダの 4 バイトを足したところです。スタンダードデータは、引数のサイズ、標準コマンド、引数で 1 つのコマンドを表し、このセットが必要回数繰り返されます。サイズが 0 (すなわち 0x0000) でスタンダードデータの終了を表します。

10.3 標準データコマンドの定義

標準データ中で描画ファンクションを表すコマンドは、sysdata.h で以下のように定義されています。

```
#define STD_SIZE        (TINY)0     /* 描画領域の指定 */
#define STD_SETPEN      (TINY)1     /* ペンの設定 */
#define STD_MOVEPEN     (TINY)2     /* ペンの移動 */
#define STD_PSET        (TINY)3     /* 点の描画 */
#define STD_LINE        (TINY)4     /* 直線の描画 */
#define STD_FRAME       (TINY)5     /* 四角形の描画 */
#define STD_BOX         (TINY)6     /* 中を塗りつぶした四角形の描画 */
#define STD_ROUND       (TINY)7     /* 角の丸い四角形の描画 */
#define STD_FILLROUND   (TINY)8     /* 中を塗りつぶした[illegible]角形の描画 */
#define STD_OVAL        (TINY)9     /* 円の描画 */
#define STD_FILLOVAL    (TINY)10    /* 中を塗りつぶした円の描画 */
#define STD_ARC         (TINY)11    /* 円弧の描画 */
#define STD_PAI         (TINY)12    /* 扇型の描画 */
#define STD_FILLPAI     (TINY)13    /* 中を塗りつぶした扇型の描画 */
#define STD_POLYGON     (TINY)14    /* 多角形の描画 */
#define STD_FILLPOLYGON (TINY)15    /* 中を塗りつぶした多角形の描画 */
#define STD_DICON       (TINY)16    /* アイコンパターンの描画 */
#define STD_WRITEBIT    (TINY)17    /* カラーデータの描画 */
#define STD_SETFONT     (TINY)18    /* フォントの指定 */
#define STD_DFONT       (TINY)19    /* 1文字表示 (1バイト文字) */
#define STD_DKANJI      (TINY)20    /* 1文字表示 (シフト JIS) */
#define STD_DSTR        (TINY)21    /* 文字列の表示 */
#define STD_TEXT        (TINY)22    /* TEXT構造体 */
#define STD_ARROW       (TINY)23    /* 矢印の描画 */
#define STD_DPATTERN    (TINY)24    /* ビットマップパターンの描画 */
#define STD_COLICON     (TINY)25    /* カラーアイコンパターンの描画 */
```

10.4 標準データのコマンド

ここでは、標準データ内に記録するコマンドについて説明します。

10.4.1 表記法

標準データのフォーマットは、次のように表記します。

コマンドの機能を示します。

コマンド	コマンド名（定義された名前）です。実際には、1バイトの数値です。
引　数	コマンドに続くデータです。それぞれのコマンドの後には、表記されているデータが続けて記録されます。
解　説	コマンドの説明です。
参　照	参照すべきマネージャやファンクションを示します。

描画領域の指定

コマンド STD_SIZE

引数

int xs 縦方向のサイズ
int ys 横方向のサイズ

解説 描画領域のサイズを指定します。

ペンの設定

コマンド STD_SETPEN

引数 PEN pen PEN構造体

解説 描画用のペンを設定します。

参照 PEN構造体（グラフパック）

ペンの移動

コマンド STD_MOVEPEN

引数

int x 縦方向の位置
int y 横方向の位置

解説 x、yで指定した位置にペンを移動します。

参照 _movepen()（グラフパック）

点の描画

STD_PSET

引数　なし

解説　現在のペンの位置に点を表示します。

参照　_pset()（グラフパック）

直線の描画

コマンド　STD_LINE

引数
int x 縦方向の位置
int y 横方向の位置

解説　現在のペンの位置からx、yで指定した位置まで直線を描画します。

参照　_line()（グラフパック）

四角形の描画

コマンド　STD_FRAME

引数
int x 縦方向の位置
int y 横方向の位置

解説　現在のペンの位置とx、yを対角線とする四角形を描画します。

参照　_frame()（グラフパック）

中を塗りつぶした四角形の描画

コマンド	STD_BOX
引 数	int x 縦方向の位置 int y 横方向の位置
解 説	現在のペンの位置と x、y を対角線とする、中を塗りつぶした四角形を描画します。
参 照	_box() (グラフパック)

角の丸い四角形の描画

コマンド	STD_ROUND
引 数	int x 縦方向の位置 int y 横方向の位置
解 説	現在のペンの位置と x、y を対角線とする、角の丸い四角形を描画します。
参 照	_round() (グラフパック)

中を塗りつぶした角の丸い四角形の描画

コマンド	STD_FILLROUND
引 数	int x 縦方向の位置 int y 横方向の位置
解 説	現在のペンの位置と x、y を対角線とする、中を塗りつぶした角の丸い四角形を描画します。
参 照	_fillround() (グラフパック)

円の描画

コマンド　STD_OVAL

引 数　int x 縦方向の位置
int y 横方向の位置

解 説　現在のペンの位置と x、y を対角線とする四角形に内接する円を描画します。

参 照　_oval() (グラフパック)

中を塗りつぶした円の描画

コマンド　STD_FILLOVAL

引 数　int x 縦方向の位置
int y 横方向の位置

解 説　現在のペンの位置と x、y を対角線とする四角形に内接する、中を塗りつぶした円を描画します。

参 照　_filloval() (グラフパック)

円弧の描画

コマンド	STD_ARC

引数

int	x	縦方向の位置
int	y	横方向の位置
TINY	startangle	開始角度
TINY	endangle	終了角度

解説

startangle で指定された角度から endangle で指定された角度までの円弧を時計回りに描画します。図形は現在のペンの位置と、x、y で指定した位置を対角線とする四角形に内接する円の一部となります。

参照

_arc()（グラフパック）

扇型の描画

コマンド	STD_PAI

引数

int	x	縦方向の位置
int	y	横方向の位置
TINY	startangle	開始角度
TINY	endangle	終了角度

解説

startangle で指定された角度から endangle で指定された角度までの扇型を時計回りに描画します。図形は現在のペンの位置と x、y で指定した位置を対角線とする四角形に内接する円の一部となります。

参照

_pai()（グラフパック）

中を塗りつぶした扇型の描画

コマンド	STD_FILLPAI
引数	int x 縦方向の位置 int y 横方向の位置 TINY startangle 開始角度 TINY endangle 終了角度
解説	startangle で指定された角度から endangle で指定された角度までの、中を塗りつぶした扇型を時計回りに描画します。図形は現在のペンの位置と、x、y で指定した位置を対角線とする四角形に内接する円の一部となります。
参照	_fillpai() (グラフパック)

多角形の描画

コマンド	STD_POLYGON
引数	TINY num 多角形の頂点の数 POS pos[] 頂点座標の配列 (可変長)
解説	頂点座標の配列にしたがって、num個の頂点を持つ多角形を描画します。
参照	_polygon() (グラフパック)

中を塗りつぶした多角形の描画

コマンド	STD_FILLPOLYGON
引数	TINY **num** 多角形の頂点の数 POS pos[] 頂点座標の配列（可変長）
解説	頂点座標の配列にしたがって、num個の頂点を持つ、中を塗りつぶした多角形を描画します。
参照	_fillpolygon()（グラフパック）

ビットマップデータの描画

コマンド	STD_DICON
引数	char pat[] ビットマップデータ（可変長）
解説	pat[]で指定されるビットマップデータを描画します。データは1ドットが1ビットで表されます。
参照	_dicon()（グラフパック）

カラーコードの描画

コマンド	STD_WRITEBIT
引数	AREA area 描画エリア COLOR color[] カラーコードの配列（可変長）
解説	colorで指定されるカラーコードの配列をareaで指定した領域に描画します。1ドットが1バイトで表されます。
参照	writebit()（グラフパック）

フォントの設定

コマンド	STD_SETFONT
引　数	FONT　font　FONT構造体
解　説	文字表示用のフォントを設定します。
参　照	FONT構造体（フォントパック）

1文字表示（1バイト文字）

コマンド	STD_DFONT
引　数	char　c　表示する文字
解　説	1バイトコードで表される、英数記号文字を表示します。
参　照	_dfont()（フォントパック）

1文字表示（シフトJIS）

コマンド	STD_DKANJI
引　数	WORD　sjis　シフトJIS文字コード
解　説	sjisによって指定した文字を表示します。
参　照	dkanji()（フォントパック）

文字列の表示

コマンド	STD_DSTR
引　数	`char str[]` 文字列（可変長）
解　説	文字列を表示します。文字列の最後には0x00が必要です。
参　照	_dstr()（フォントパック）

TEXTの設定と表示

コマンド	STD_TEXT
引　数	`TEXT text` TEXT構造体
解　説	textで指定したテキストの設定と表示を行います。
参　照	TEXT構造体（テキストマネージャ）

矢印の設定と表示

コマンド	STD_ARROW
引　数	`ARROWINFO arrowinfo` ARROWINFO構造体
解　説	arrowinfoで指定した設定で矢印を表示します。

フォントの設定にしたがったパターンの表示

コマンド	STD_DPATTERN
引数	char pat[] パターン（可変長）
解説	現在のフォントの設定にしたがった飾りつけをして、pat[ ] で指定したパターンを表示します。パターンのサイズは現在のフォントと同一でなければなりません。
参照	_dpattern()（フォントパック）

カラーパターンの表示

コマンド　STD_COLICON

引数

WORD	xsize	横方向のサイズ
WORD	ysize	縦方向のサイズ
COLOR	oncolor	オンドットの色
COLOR	offcolor	オフドットの色
char	pat[]	パターン（可変長）

解説　pat[ ] で指定されるパターンを xsize、ysize の大きさと、oncolor、offcolor の色で表示します。

参照　_colicon()（グラフパック）

11章 ディスプレイマネージャ

この章では、ディスプレイマネージャの構成や各ファンクションについて説明します。

11.1 ディスプレイマネージャとは

MSXViewでの画面表示はすべて、VDP（V9958）のビットマップグラフィックスモードを使用したオーバーラップウィンドウ（互いに重なりあうことのできるウィンドウ）環境で行われます。ディスプレイマネージャは、このオーバーラップウィンドウ環境を統一的に管理する重要なマネージャで、VDPの4種類すべてのビットマップグラフィックスモードを使用することができます。

MSXViewでは、ウィンドウ環境を維持するために、画面への表示は必ずディスプレイマネージャをとおして行います。一般的なオーバーラップウィンドウ環境(例えば、MacOSやMS-Windowsなど）では、他のウィンドウによって隠されたウィンドウを表に出すためには、アプリケーションがウィンドウを再描画しなければなりません。しかし、MSXViewのウィンドウ環境では、ウィンドウの重なりにより失われる部分は、ディスプレイマネージャマネージャが自動的に待避・回復するので、アプリケーション側で、再描画する必要はありません。ただし、重ねられるウィンドウの総面積は、裏VRAM1画面分までに制限されます。このウィンドウを、「FLOATウィンドウ」と呼びます。

FLOATウィンドウは、画面上に開ける面積に限りがあるため、MSXViewには、「FIXウィンドウ」というオーバーラップができないウィンドウもあります。各アプリケーションの描画領域、作業領域など大きな面積を使用する部分をウィンドウ管理するには、FIXウィンドウが便利です。FIXウィンドウは重なりあうことができませんが、FIXウィンドウとFLOATウィンドウとが重なりあうことはできます。

11.2 ディスプレイマネージャの使い方

画面に何かを描くときは、以下の手順で処理を進めます。

1. 描画するウィンドウをカレントウィンドウにする。
   まず、描画するウィンドウをアクティブにします。画面上で同時にアクティブになることのできるウィンドウは1つだけで、MSXViewではこれをカレントウィンドウと呼びます。カレントウィンドウを変更するには、_chwin()、_pushwin() などのファンクション（後述）を使います。
2. 描画するエリアをズームする。
   描画するためには、描画するエリアを指定しなければなりません。これを、MSXViewではズームと呼びます。ズームすることにより、以下のような処理が行われます。
   - ズームされる描画領域の上に別のウィンドウがあるときは、そのウィンドウの重なっている部分だけが一時的に取り除かれ描画領域が表面に表示されます。
   - ズームされた描画領域にカーソルが重なる場合には、カーソルを表示しないようにします。
3. グラフパック、フォントパックなどを使用して描画する。
   図形（文字）を描く場合はすべて、ズームした描画領域に対して行われます。MSXViewでは様々な図形や文字を描くために、グラフパックおよびフォントパックが用意されています。
4. エンドズームして、描画環境を回復する。
   描画が終わったらできる限りすみやかにズーム状態を解いて下さい。ズーム状態では、描画領域に重なったカーソルが表示されないなどの問題があるので、描画が終わったらすみやかに_endzoom() ファンクションを呼んで、描画環境を回復して下さい。
5. ウィンドウ環境を元に戻す。
   描画のためにカレントウィンドウを変更したときには、これを元に戻すようにつとめて下さい。ウィンドウ環境を元に戻さないと、後の描画処理に問題が起こることがあります。

画面に何か図形を描くときは、ウィンドウ上の描画領域をズームし、グラフパック、フォントパックを使って書き込んで行きます。ズーム中は描画領域内では、カーソルは自動的に消えます。書き込みが終了したら、必ずエンドズームして、描画環境を元に戻して下さい。

ウィンドウ間を図形が横切るような特別の場合（ウィンドウの移動など）は、ダイレクトにルートボードに書き込むこともできますが、ラバーバンド（11.3.1「ルートボード」参照）を使うなどして、画面を壊さないように注意して下さい。

11.3 ディスプレイマネージャの構成

ディスプレイマネージャは、以下のような概念で画面を管理します。

11.3.1 ルートボード

ルートボードとは、ウィンドウが配置される画面そのもので、画面と同じサイズの特殊なウィンドウです。画面上で動くことのない情報などを表示します。ウィンドウを移動するときは、_setrub() を使って、ルートボード上にラバーバンドで枠を表示することができます。ルートボードは、色とタイルパターンを持つことができます。MSXView では、ルートボードも1つのウィンドウとして扱うようになっており、ウィンドウハンドル1が割り当てられています。

11.3.2 ウィンドウ

ウィンドウには「FIX ウィンドウ」と「FLOAT ウィンドウ」の2種類があります。「FIX ウィンドウ」には、「FLOAT ウィンドウ」にはない制限があります。

FIX ウィンドウ

FIX ウィンドウとは、待避画面エリアを持たないウィンドウのことです。このウィンドウには、画面を保存する領域がないので、クローズしたときは画面の情報が消えてしまいます。FIX ウィンドウどうしを重ねることはできませんが、画面上で重ならない限り、画面一杯まで開くことができます。

FLOAT ウィンドウ

FLOAT ウィンドウと違い、制限はありません。自由に重ね合うことができ、移動させることもできます。ただし、開ける FLOAT ウィンドウの面積は、合計で画面1枚分までです。

表 3.4 FLOAT ウィンドウの種類

名前	形
TWIN	何もなし（特別な用途のための透明なウィンドウ）
CWIN	枠なし
FWIN	枠つき
RWIN	角の丸い枠つき
IWIN	インデックス（上端の角が丸い枠）つき
SWIN	影付き（右下に影がつく）
BWIN	二重枠つき

FWIN RWIN IWIN SWIN BWIN

図 3 2 各ウィンドウの形状

通常の描画領域にはFWINを、ポップアップメニューなど注目させたい部分には、SWINまたはBWINを使うの一般的です。

それぞれのウィンドウがどのように定義されているかは、9.3.1「ウィンドウスタイルの定数」を参照して下さい。

11.3.3 ズームの概念

ディスプレイマネージャにおけるズームとは、ディスプレイマネージャに描画を宣言することです。ズームされた領域は、たとえ他のウィンドウで覆い隠されていても、一時的に表に出されます。このため、ウィンドウの優先度などを変更するときに、アプリケーションがウィンドウを書き直すなどの手間がかかりません。また同時に、マウスカーソルがズームエリアに重なる場合には、自動的にマウスカーソルも消去されて、描画の邪魔をしないようにします。MSXView環境では、マウスカーソルもウィンドウ上のさまざまな表示も同じビットマップVRAMに格納されるので、ズームを行なわずに画面をアクセスすると、いろいろな障害が生じてしまいます。

ズームすると、オーバーラップウィンドウ環境が一時的に破壊されます。したがって、ズームは必要最小限の領域を、必要最小限の時間だけ行なう（描画が終了したら直ちに_endzoom()を呼ぶ）ようにして下さい。そうしないと、マウスカーソルが消えるなどの問題が発生します。

11.3.4 ディスプレイマネージャで使用するデータ構造

ディスプレイマネージャではウィンドウを管理するために次のような構造体を使用しています。

```
typedef struct _win {
        HANDLE  block;          /* ビットブロックハンドル */
        TINY    status;         /* ウィンドウの状態 */
        AREA    area;
        HANDLE  defpen;         /* ウィンドウデフォルトペン */
        HANDLE  deffont;        /* ウィンドウデフォルトフォント */
        HANDLE  curpen;         /* ウィンドウカレントペン */
        HANDLE  curfont;        /* ウィンドウカレントフォント */
}       WIN;
```

statusの意味は次のようになっています。

表 3 5 statusの内容

ビット	意味
7	FIX
6	Closed
5	Clear
4	reserved
3	Window style
2	Window style
1	Windowstyle
0	Window style

11.3.5 ペンハンドルとフォントハンドル

MSXView では、描画のための環境をペン（PEN）と呼び、文字の属性を表現するためのさまざまな情報の集まりをフォントテンプレート（F●NT）と呼びます。ディスプレイマネージャは、PEN や FONT を、ハンドルを通して扱うための機能を持っています。これらの情報をハンドル管理ことにより、以下のようなメリットがあります。

- PEN、FONT などはサイズの大きな構造体なので、ハンドルで管理するとメモリ効率が良い。
- 複数のモジュールで共通の PEN、FONT を簡単に共用することができる。
- 異なるウィンドウでは、描画に使用するペンやフォントの環境が大きく異なることが多いが、PEN や FONT をウィンドウの属性として持たせておくことで、ウィンドウを切り換えるだけで、PEN や FONT も自動的に切り換えられる。

PEN や FONT の詳細については、グラフパックおよびフォントパックを参照して下さい。

11.3.6 デフォルトとカレント

ペンハンドルとフォントハンドルには、それぞれカレントペン、カレントフォントが1つだけ存在します。同時に複数のペンやフォントがカレントとなることはありません。グラフパックはカレントペンに基づいて描画し、フォントパックはカレントフォントに基づいて、文字を表示します。

カレントペンを変更するには、_chpen()、_pushpen() などのファンクションを使用します。また カレントフォントを変更するには、_chfont()、_pushfont() などのファンクションを使用します。

これとは別に、デフォルトペン、デフォルトフォントというものが、ウィンドウごとに存在します。これは、ウィンドウを作成したときに指定したペンとフォントのことで、_chpen(0)、_chfont(0) などの簡単な手続きで、カレントペンやカレントフォントをデフォルトペン、デフォルトフォントに戻すことができます。そのウィンドウで、最も使用頻度の高いペンやフォントをデフォルトにしておくと便利です。

11.4 ファンクション一覧

ディスプレイマネージャには、以下のファンクションがあります。

表 3.6 ディスプレイマネージャのファンクション一覧

機能番号	名前	意味	ページ
2	_screen()	スクリーンモードの設定	406
47	_initpenhd()	ペンハンドルの初期化	399
48	_createpen()	ペンの作成	400
49	deletepen()	ペンの削除	401
50	_chpen()	カレントペンの変更	402
51	_currentpen()	カレントペンの獲得	402
52	_pushpen()	保存をともなうカレントペンの変更	403
53	_poppen()	カレントペンの復帰	403
54	_penadrs()	ペン情報の獲得	404
55	_renewpen()	ペンの更新	405
56	_getdefpen()	デフォルトペンの獲得	406
57	_initfonthandle()	フォントハンドルの初期化	400
58	_createfont()	フォントの作成	401
59	_deletefont()	フォントの削除	401
60	_chfont()	カレントフォントの変更	402
61	_currentfont()	カレントフォントの獲得	403
62	_pushfont()	保存をともなうカレントフォントの変更	404
63	_popfont()	フォントの復帰	404
64	_fontadrs()	フォント情報の獲得	405
65	_renewfont()	フォントの更新	405
66	getdeffont()	デフォルトフォントの獲得	406
68	_clearrootbd()	ルートボードのクリア	399
71	_initwin()	ウィンドウマネージャの初期化	389
72	_createwin()	ウィンドウの作成	390
73	_openwin()	ウィンドウのオープン	391
74	_closewin()	ウィンドウのクローズ	391
75	_deletewin()	ウィンドウの削除	392

機能番号	名前	意味	ページ
76	_clearwin()	ウィンドウのクリア	392
77	_getwininfo()	ウィンドウ情報の獲得	392
78	_setwininfo()	ウィンドウの変更	393
79	_frontwin()	ウィンドウを最前面に移動	393
80	_backwin()	ウィンドウを最後面に移動	394
81	_movewin()	ウィンドウの位置の移動	394
82	resizewin()	ウィンドウサイズの変更	395
83	_findwin()	ウィンドウハンドルの獲得	395
84	chwin()	カレントウィンドウの変更	396
85	_currentwin()	カレントウィンドウの獲得	396
86	_pushwin()	保存をともなうカレントウィンドウの変更	396
87	_popwin()	カレントウィンドウの復帰	397
88	_gtol()	グローバル座標からローカル座標への変換	397
89	ltog()	ローカル座標からグローバル座標への変換	397
90	_movepopup()	ポップアップウィンドウ位置の設定	399
91	_zoom()	エリアのズーム	398
92	_zoomwin()	ウィンドウのズーム	398
93	_endzoom()	ズームの終了	398
374	_screensize()	スクリーンサイズの獲得	407
388	_getscreenmode()	スクリーンモードの獲得	407

11.5 ファンクションの説明

以下では、ディスプレイマネージャの各ファンクションについて説明します。

11.5.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号 各ファンクションに割り当てられている番号です。

書 式 各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- ファンクションの戻り値の型
- ファンクション名
- 引数
- 引数の意味
- 引数の受け渡しに使われるCPUレジスタ(アセンブラプログラミング時に使用)
- 引数名
- 引数の型

戻り値 そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- 戻り値の意味
- 戻り値
- 戻り値の引き渡しに使われるCPUレジスタ(アセンブラプログラム時に使用)

解 説 そのファンクションがどのような動作をするかを示します。

ウィンドウマネージャの初期化

機能番号	71
書式	void _initwin(void)
戻り値	なし
解説	ウィンドウマネージャを初期化します。ビットブロックおよびズームもクリアします。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

ウィンドウの作成

機能番号 72

書　式

```
HANDLE   createwin(area, style, win, pen, font)
AREA     *area  [HL]  ウィンドウの領域
TINY     style  [E]   ウィンドウの形状
                      (FIXまたはFLOATの指定)
HANDLE   win    [C]   ウィンドウハンドル
HANDLE   pen    [A']  デフォルトとなるペンのハンドル
HANDLE   font   [E']  デフォルトとなるフォントのハンドル
```

戻り値 [A] ウィンドウハンドル
作成できなかった場合ERROR

解　説 新しいウィンドウを作成します。

areaには、AREA構造体(9.2「基本的な構造体」参照)へのポインタを指定します。

styleには、FIXウィンドウやFLOATウィンドウのスタイルを指定します。ウィンドウスタイルについては、9.3.1「ウィンドウスタイルの定数」および11.3.2「ウィンドウスタイル」を参照して下さい。

winが0の場合は新しいハンドルを割り付け、それ以外の場合はその値をハンドルとして割り付けます。このとき、複数のウィンドウが1つのハンドルを共有することはできないので、アプリケーション内でウィンドウを固定的に割り付けるなどの特殊な場合を除き、通常はwinに0をセットしてコールして下さい。同じ値を持ったウィンドウハンドルが存在するときは、すでに存在しているウィンドウが削除され、新しく作られたものに置き換えられます。ウィンドウハンドル1は、ルートボードとして割り当てられているため、通常のウィンドウとしては使用できません。

penとfontには、そのウィンドウのデフォルトとなるペンとフォントを指定します。このとき、penにSYSPEN、fontにはSYSFONT(共に値は1)を指定すると、システムの標準ペンと標準フォントが使用されます。NEW(値は0)を指定した場合には、新しいハンドルが割り当てられ、自動的に標準ペン、標準フォントの内容がコピーされます。この場合、割り付けられたハンドルを知るには、後述の_getdefpen()、_getdeffont()を使用します。

ウィンドウのオープン

機能番号　73

書式

```
STATUS  _openwin(win, x, y)
HANDLE  win  [A]   ウィンドウハンドル
WORD    x    [BC]  横方向のグローバル位置
WORD    y    [DE]  縦方向のグローバル位置
```

戻り値

```
[A] OK     オープン成功
    ERROR  オープン失敗
```

解説

win で指定したウィンドウを、x、y を左上とする位置にオープンします。x、y 両方に 0xffff を指定すると、作成されたときの位置（デフォルトの位置）にオープンされます。

ウィンドウのクローズ

機能番号　74

書式

```
STATUS  _closewin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値

```
[A] OK     クローズ成功
    ERROR  クローズ失敗
```

解説

win で指定したウィンドウをクローズします。画面からは消去されますが、裏画面にある保存領域はそのままなので、もう一度、_openwin() を使用してオープンすれば前の状態で画面上に表示されます。本当にそのウィンドウが不必要になったときは、_deletewin() を実行して下さい。ただし、FIX ウィンドウの場合は保存領域がないので、もう一度オープンするとウィンドウはクリアされた状態になってしまいます。

ウィンドウの削除

機能番号 75

書式

```
STATUS  _deletewin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値

```
[A]  OK     削除成功
     ERROR  削除失敗
```

解説

winで指定したウィンドウを削除します。オープンされているウィンドウが指定されると、クローズしてから削除されます。また、裏画面に確保されている保存領域も同時に解放されます。

ウィンドウのクリア

機能番号 76

書式

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値

```
[A]  OK     クリア成功
     ERROR  クリア失敗
```

解説

winで指定したウィンドウをクリアします。ウィンドウが新しく作成されたときと同じ表示になります。

ウィンドウ情報の獲得

機能番号 77

書式

```
STATUS  _getwininfo(win, info)
HANDLE  win    [A]   ウィンドウハンドル
WIN     *info  [DE]  WIN構造体へのポインタ
```

戻り値

```
[A]  OK     獲得成功
     ERROR  獲得失敗
```

解説

winで指定したウィンドウの状態をinfoに返します。

ウィンドウの変更

機能番号　78

書　式
```
STATUS  _setwininfo[illegible], info)
HANDLE  win    [A]   ウィンドウハンドル
WIN     *info  [OE]  WIN構造体へのポインタ
```

戻り値
```
[A]  OK     変更成功
     ERROR  変更失敗
```

解　説　winで指定したウィンドウの状態を、infoに設定した内容にしたがって変更します。このファンクションでは、アプリケーションでは使用しません。

ウィンドウを最前面に移動

機能番号　79

書　式
```
STATUS  _frontwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値
```
[A]  OK     移動成功
     ERROR  移動失敗
```

解　説　画面上に複数のウィンドウがオーバーラップして表示されているときに、winで指定したウィンドウを最前面に表示します。

ウィンドウを最後面に移動

機能番号　80

書　式
```
STATUS  _backwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値
```
[A]  OK      移動成功
     ERROR   移動失敗
```

解　説

画面上に複数のウィンドウがオーバーラップして表示されているときに、win で指定したウィンドウを最後面に表示します。

ウィンドウの位置の移動

機能番号　81

書　式
```
STATUS  _movewin(win, xp, yp)
HANDLE  win  [A]   ウィンドウハンドル
WORD    xp   [DE]  横方向のグローバル座標位置
WORD    yp   [BC]  縦方向のグローバル座標位置
```

戻り値
```
[A]  OK      移動成功
     ERROR   移動失敗
```

解　説

win で指定したウィンドウを、xp、ypで指定した位置を左上とするグローバル座標へ移動します。FIX ウィンドウ同士の重なりなどはチェックしません。

ウィンドウサイズの変更

機能番号　82

書　式

```
STATUS  _resizewin(win, xsize, ysize)
HANDLE  win    [A]   ウィンドウハンドル
WORD    xsize  [BC]  横方向のサイズ
WORD    ysize  [DE]  縦方向のサイズ
```

戻り値　[A]　OK　変更成功
　　　　　　ERROR　変更失敗

解　説

winで指定したウィンドウを、xsize、ysizeで指定したサイズに変更します。ウィンドウを縮小したときは右下方向がカットされ、拡大したときは右下方向に広がります。

ウィンドウハンドルの獲得

機能番号　83

書　式

```
HANDLE  _findwin(where)
POS     *where  [HL]  グローバル座標位置
```

戻り値　[A]　ウィンドウハンドル
　　　　　　どのウィンドウにも属していない場合ERROR

解　説

whereで与えられたグローバル座標にオープンされているウィンドウのハンドルを返します。このファンクションは、マウスクリックなどで　マウスカーソルがどのウィンドウの上にいるのかを調べるのに使います。

カレントウィンドウの変更

機能番号 84

書式
```
STATUS  _chwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値 [A] OK 変更成功
ERROR 変更失敗

解説 カレントウィンドウをwinで指定したウィンドウに変更します。_gtol() _zoom()などをはじめ、描画に関するすべての処理は、カレントウィンドウに対して行われます。

カレントウィンドウの獲得

機能番号 85

書式
```
HANDLE  _currentwin(void)
```

戻り値 [A] カレントウィンドウのハンドル

解説 カレントウィンドウのハンドルを返します。

保存をともなうカレントウィンドウの変更

機能番号 86

書式
```
STATUS  _pushwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

戻り値 [A] OK 変更成功
ERROR 変更失敗

解説 カレントウィンドウをスタックに保存し、winで指定したウィンドウをカレントに変更します。_popwin()を使うと、カレントウィンドウを元に戻すことができます。

カレントウィンドウの復帰

機能番号 87

書　式

```
STATUS  popwin(void)
```

戻り値 [A] OK　復帰成功
ERROR　復帰失敗

解　説 スタックに保存したウィンドウハンドルを取り出し、カレントウィンドウとします。_pushwin() と対にして使います。

グローバル座標からローカル座標への変換

機能番号 88

書　式

```
POS  *_gtol(global, local)
POS  *global  [HL]  グローバル座標
POS  *local   [DE]  ローカル座標
```

戻り値 [HL]　local へのポインタ

解　説 global で指定したグローバル座標の1点を、カレントウィンドのローカル座標へ変換して、local に返します。ウィンドウ内でのマウスカーソルの位置などを調べるために使います。

ローカル座標からグローバル座標への変換

機能番号 89

書　式

```
POS  *_ltog(local, global)
POS  *local   [HL]  ローカル座標
POS  *global  [HL]  グローバル座標
```

戻り値 [HL]　global へのポインタ

解　説 local で指定されたローカル座標の1点を、グローバル座標に変換して global に返します。

エリアのズーム

機能番号　91

書式
```
void  _zoom(area)
AREA  *area  [HL]  ズームするエリア
```

戻り値　なし

解説　カレントウィンドウの指定された領域を、描画できる状態にします。指定した領域が、他のウィンドウの下に隠されているときは、1番上に出します。その領域では、マウスカーソルは表示されません。

ウィンドウのズーム

機能番号　92

書式
```
void  _zoomwin(void)
```

戻り値　なし

解説　カレントウィンドウ全体を、描画できる状態にします。ウィンドウを開いた直後の描画などに便利ですが、マウスカーソルがウィンドウに少しでも重なっていると、マウスカーソルが消えてしまうので、ウィンドウを開いた直後の初期化のための描画など、大きなエリアを書き換えるときに使います。

ズームの終了

機能番号　93

書式
```
void  _endzoom(void)
```

戻り値　なし

解説　ウィンドウの重なりを元に戻し、カーソル消去を解除します。_zoom()、_zoomwin()をコールした後、描画が終了したら、できる限りすみやかにこのファンクションを実行して下さい。

ルートボードのクリア

機能番号	68
書式	void _clearrootbd(void)
戻り値	なし
解説	ルートボードをクリアします。

ポップアップウィンドウ位置の設定

機能番号　90

書式

```
void  _movepopup(center, area)
POS   *center  [HL]  中心座標
AREA  *area    [DE]  ポップアップウィンドウを表示するエリア
```

戻り値　なし

解説

ポップアップウィンドウを表示する位置を設定するためのファンクションです。center にポップアップの中心座標、area にウィンドウの大きさを指定すると、自動的に center を中心とし、画面に収まるようにポップアップウィンドウの位置を修正します。このファンクションは、メニューマネージャがポップアップメニューの位置を決定するときにも使われています。

ペンハンドルの初期化

機能番号	47
書式	void _initpenhd(void)
戻り値	なし
解説	ペンハンドルの初期化を行ないます。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

フォントハンドルの初期化

機能番号	57
書式	`void _initfonthandle(void)`
戻り値	なし
解説	フォントハンドルを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

ペンの作成

機能番号	48
書式	HANDLE createpen(peninfo, pen) PEN *peninfo [HL] PEN構造体へのポインタ HANDLE pen [E] ペンハンドル
戻り値	[A] 作成されたペンのハンドル 作成に失敗した場合ERROR
解説	ペンを作成します。peninfoがNULLのときは、カレントウィンドウのデフォルトペンがコピーされます。penが0のときは、新しいハンドルを割り付けてこれを返します。

フォントの作成

機能番号　58

書式

```
HANDLE  _createfont(info, font)
FONT    *info  [HL]  FONT構造体へのポインタ
HANDLE  font   [E]   フォントハンドル
```

戻り値　[A]　作成されたフォントのハンドル
　　　　　　　作成できなかった場合ERROR

解説　フォントを作成し、そのハンドルを返します。infoが0のときは、システムフォントの設定がコピーされます。fontが0のときは、新しいフォントハンドルを返します。

ペンの削除

機能番号　49

書式

```
void    _deletepen(pen)
HANDLE  pen  [A]  ペンハンドル
```

戻り値　なし

解説　penで指定したペンを削除します。削除するのはアプリケーションで作成したペンだけにして下さい。SYSPENなどを削除すると、MSXViewが正常に動作しなくなります。

フォントの削除

機能番号　59

書式

```
STATUS  _deletefont(font)
HANDLE  font  [A]  フォントハンドル
```

戻り値　[A]　OK　　削除成功
　　　　　　　ERROR　削除失敗

解説　fontで指定したフォントを削除します。

カレントペンの変更

機能番号 50

書式
```
void    _chpen(pen)
HANDLE  pen  [A]  変更するペンのハンドル
```

戻り値 なし

解説 カレントウィンドウに属するカレントペンを、penで指定したペンに変更します。penが0のときは、カレントウィンドウのデフォルトペンが設定されます。

カレントフォントの変更

機能番号 60

書式
```
void    _chfont(font)
HANDLE  font  [A]  変更するフォントのハンドル
```

戻り値 なし

解説 カレントウィンドウに属するカレントフォントを、fontで指定したフォントに変更します。fontが0のときは、カレントウィンドウのデフォルトフォントが設定されます。

カレントペンの獲得

機能番号 51

書式
```
HANDLE  _currentpen(void)
```

戻り値 [A] ペンハンドル

解説 カレントペンのハンドルを返します。

カレントフォントの獲得

機能番号	61
書式	HANDLE _currentfont(void)
戻り値	[A] フォントハンドル
解説	カレントフォントのハンドルを返します。

保存をともなうカレントペンの変更

機能番号	52
書式	STATUS pushpen(pen) HANDLE pen [A] ペンハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗
解説	カレントペンをスタックに保存し、pen で指定したペンをカレントとします。_poppen() を使うと、カレントペンを元に戻すことができます。pen が 0 のときは、カレントウィンドウのデフォルトペンが設定されます。

カレントペンの復帰

機能番号	53
書式	STATUS _poppen(void)
戻り値	[A] OK 復帰成功 ERROR 復帰失敗
解説	スタックに保存したペンハンドルを取り出し、カレントペンを元に戻します。_pushpen() と対にして使います。

保存をともなうカレントフォントの変更

機能番号 62

書式

```
STATUS  _pushfont(font)
HANDLE  font  [A]  フォントハンドル
```

戻り値 [A] OK 変更成功
ERROR 変更失敗

解説 カレントフォントをスタックに積み、fontで指定したフォントをカレントフォントとします。fontが0のときは、ウィンドウのデフォルトフォントが設定されます。

フォントの復帰

機能番号 63

書式

```
STATUS  _popfont(void)
```

戻り値 [A] OK 復帰成功
ERROR 復帰失敗

解説 スタックに保存したフォントハンドルを取りだし、カレントフォントを元に戻します。_pushfont()と対にして使います。

ペン情報の獲得

機能番号 54

書式

```
PEN     *_penadrs(pen)
HANDLE  pen  [A]  ペンのハンドル
```

戻り値 [HL] PEN構造体の先頭アドレス

解説 penで指定したPEN構造体のアドレスを返します。PENの内容を一部変更するときなどに使います。

フォント情報の獲得

機能番号	64
書　式	FONT　* fontadrs(font) HANDLE　font　[A]　フォントハンドル
戻り値	[HL]　FONT 構造体の先頭アドレス
解　説	font で指定した FONT 構造体のアドレスを返します。FONT の内容を一部変更するときなどに使います。

ペンの更新

機能番号	55
書　式	STATUS　_renewpen(pen) HANDLE　pen　[A]　ペンのハンドル
戻り値	[A]　OK　更新成功 ERROR　更新失敗
解　説	_penadrs() などを使って、PEN の内容を書き換えたときに、その新しいペンで描画する前に呼び出します。

フォントの更新

機能番号	65
書　式	STATUS　_renewfont(font) HANDLE　font　[A]　フォントハンドル
戻り値	[A]　OK　更新成功 ERROR　更新失敗
解　説	_fontadrs() などを使って、FONT の内容を書き換えたときに、その新しいフォントで文字を出力する前に呼び出します。_chfont() とほぼ同様の処理を行ないますが、それよりも高速です。

デフォルトペンの獲得

機能番号	56
書式	`HANDLE _getdefpen(void)`
戻り値	[A] カレントウィンドウのデフォルトペン
解説	カレントウィンドウのデフォルトペンを返します。

デフォルトフォントの獲得

機能番号	66
書式	`HANDLE _getdeffont(void)`
戻り値	[A] フォントハンドル
解説	デフォルトフォントのハンドルを返します。

スクリーンモードの設定

機能番号	2
書式	`void _screen(mode)` `TINY mode` [A] スクリーンモード番号 (BASICと同様)
戻り値	なし
解説	スクリーンモードを変更します。指定できるスクリーンモードは5~8までと、10~12までです。不正なスクリーンモードが指定されたときは何もせず戻ります。

スクリーンモードの獲得

機能番号	388
書　式	TINY _getscreenmode(void)
戻り値	[A] スクリーンモード番号（BASICと同様）
解　説	現在のスクリーンモードを返します。

スクリーンサイズの獲得

機能番号	374
書　式	void _screensize(area) AREA *area [HL] スクリーンサイズが返される AREA構造体へのポインタ
戻り値	なし
解　説	現在のスクリーンサイズをareaに返します。

12章
ビットブロックマネージャ

この章では、ビットブロックマネージャの構成や各ファンクションについて説明します。

12.1 ビットブロックマネージャとは

ビットブロックマネージャは、ウィンドウ処理を高速化するために、画面上の矩形領域を効率的に裏VRAMに格納するためのマネージャです。オーバーラップウィンドウの再描画処理をアプリケーションが行なわなくてもよいのは、ビットブロックマネージャが、隠される部分を横に1ラインずつスライスし、上から順に裏VRAMへ待避しているからです。

ビットブロックマネージャは、「ブロック」と「ロット」という概念でデータを管理します。ブロックとは、1つのデータを表すビットの集まりで、複数のブロックの集まりをロットと呼びます。

ディスプレイマネージャはMSXView起動時やスクリーンモードが変わるときにビットブロックマネージャを初期化して、SYSLOTとAPLLOTというロットを1つずつ作ります。SYSLOTはロット番号1、APLLOTはロット番号2です。

ビットブロックマネージャは、ウィンドウ管理の内部処理ルーチンとしてディスプレイマネージャが使用しますが、アプリケーションでもデータ保存領域として使うこともできます。この場合、アプリケーションはAPLLOTつまりロット番号2のみが使用可能です。ディスプレイマネージャもAPLLOTを使用するので、アプリケーションがデータ領域として使うときは、その分ウィンドウの退避領域が減ることになります。

ディスプレイマネージャやアプリケーションが、_newblc()を呼んでブロックを作成すると、ハンドルが返され、それ以降のブロックに対する読み書きは、そのハンドルでブロックを指定して行ないます。ブロックの使用が終了したら、_freeblc()を呼んでそのブロックを解放しなければなりません。

なおスクリーンモードを変更すると、画面は初期化され、すべてのデータおよびウィンドウは消去されます。

12.2 ファンクション一覧

ビットブロックマネージャには、以下のファンクションがあります。

表3.7 ビットブロックマネージャのファンクション一覧

機能番号	名前	意味	ページ
118	_initblc()	ビットブロックマネージャの初期化	412
119	_newlot()	新規ロットの割り付け	412
120	_freelot()	ロットの解放	413
121	_newblc()	新規ブロックの獲得	413
122	freeblc()	ブロックの解放	413
123	putblc()	ブロックへの保存	417
124	getblc()	ブロックから画面への表示	417
125	_swapblc()	ブロック内と画面の画像の交換	418
126	_resizeblc()	ブロックサイズの変更	418
127	_storeblc()	ブロックへの画像の保存	414
128	_restoreblc()	ブロックからの画像の取り出し	414
129	_blcpoint()	ブロック上のカラーコードの獲得	415
130	_blcpixel()	ブロックへの点の書き込み	415
131	_blcread()	ブロックからメモリへの読み込み	416
132	_blcwrite()	メモリからブロックへの書き込み	416

12.3 ファンクションの説明

以下では、ビットブロックマネージャの各ファンクションについて説明します。

12.3.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号　各ファンクションに割り当てられている番号です。

書　式　各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- STATUS ― ファンクションの戻り値の型
- _clearwin ― ファンクシ[illegible]
- win ― 引数
- ウィンドウハンドル ― 引数の意味
- [A] ― 引数の受け渡しに使われる CPU レジスタ（アセンブラプログラミング時に使用）
- win ― 引数名
- HANDLE ― 引数の型

戻り値　そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- オープン成功／オープン失敗 ― 戻り値の意味
- OK／ERROR ― 戻り値
- [A] ― 戻り値の引き渡しに使われる CPU レジスタ（アセンブラプログラム時に使用）

解　説　そのファンクションがどのような動作をするかを示します。

ビットブロックマネージャの初期化

機能番号 118

書式

```
STATUS  initblc(xsize)
WORD    xsize  [HL]  スクリーンの横方向のサイズ
```

戻り値 [A] OK 初期化成功
ERROR 初期化失敗

解説 ビットブロックマネージャを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

新規ロットの割り付け

機能番号 119

書式

```
HANDLE  _newlot(top, size, lot)
WORD    top   [HL]  保存に使用する縦方向の位置の始まり
WORD    size  [DE]  保存に使用する縦方向のサイズ
HANDLE  lot   [C]   ロットハンドル
```

戻り値 [A] 割り付けられたロットのハンドル
lotと同じロットハンドルが存在している場合ERROR
割り付けに失敗した場合ERROR

解説 topで指定したY座標（2画面目の始まりはY=256）から、sizeで指定したYサイズまでを、データ保存用のロットとして割り当て、そのロットを参照するハンドルを返します。lotが0のときは、新しいハンドルを割り付けます。特に理由のない限り、ロットのハンドルlotは0にして下さい。ロットハンドル1と2は、システムで使用するので、指定できません。

ロットの解放

機能番号	120

書式

```
STATUS  _freelot(lot)
HANDLE  lot  [A]  ロットハンドル
```

戻り値

[A]　OK　　解放成功
　　ERROR　解放失敗

解説

lotで指定したロットを解放します。

新規ブロックの獲得

機能番号	121

書式

```
HANDLE  _newblc(lot, xsize, ysize)
HANDLE  lot    [A]   ロットハンドル
WORD    xsize  [BC]  ブロックの横方向のサイズ
WORD    ysize  [DE]  ブロックの縦方向のサイズ
```

戻り値

[A]　ブロックハンドル
　　ロットの確保に失敗した場合ERROR

解説

xsize、ysizeで指定したサイズの新しいブロックを、lotで指定したロットに確保して、そのブロックを参照するハンドルを返します。

ブロックの解放

機能番号	122

書式

```
STATUS  _freeblc(block)
HANDLE  block  [A]  ブロックのハンドル
```

戻り値

[A]　OK　　解放成功
　　ERROR　解放失敗

解説

blockで指定した番号のブロックを解放し、そこを参照するハンドルを破棄します。

ブロックへの画像の保存

機能番号	127

書　式	

```
HANDLE  _storeblc(area, lot)
AREA    *area  [HL]  画面上のエリア
HANDLE  lot    [E]   ロットハンドル
```

戻り値	

[A]　ブロックハンドル
　　保存領域が足りなかった場合 0

解　説	

areaで指定した領域と同じサイズの新しいブロックを、lotで指定したロットに確保し、area内の画像を保存して、そのブロックを参照するハンドルを返します。戻り値が0のときは、保存領域が足りないことを示しているので、新しい画像は保存できません。

ブロックからの画像の取り出し

機能番号	128

書　式	

```
STATUS  _restoreblc(area, block)
AREA    *area  [HL]  画面上の領域
HANDLE  block  [E]   ブロックハンドル
```

戻り値	

[A]　OK　　　成功
　　ERROR　失敗

解　説	

areaで指定した領域に、blockで参照されるブロックから画面上に絵を再表示し、保存領域を解放した後にハンドルを破棄します。

ブロック上のカラーコードの獲得

機能番号 129

書式

```
COLOR   _blcpoint(block, xp, yp)
HANDLE  block  [A]   ブロックハンドル
WORD    xp     [BC]  ブロック内の横方向の座標
WORD    yp     [DE]  ブロック内の縦方向の座標
```

戻り値 [A] カラーコード

解説 blockで指定したブロック中の、xp、ypで指定した座標のカラーコードを返します。

ブロックへの点の書き込み

機能番号 130

書式

```
void    _blcpixel(block, xp, yp, color)
HANDLE  block  [A]   ブロックハンドル
WORD    xp     [BC]  ブロック内の横方向の座標
WORD    yp     [DE]  ブロック内の縦方向の座標
COLOR   color  [A']  カラーコード
```

戻り値 なし

解説 blockで指定したブロック中のxp, ypの位置に、colorで指定したカラーコードの点を書き込みます。

ブロックからメモリへの読み込み

機能番号 131

書式

```
STATUS  _blcread(block, area, buff)
HANDLE  block  [A]   ブロックハンドル
AREA    *area  [DE]  ブロック内の領域
char    *buff  [BC]  転送先アドレス
```

戻り値

```
[A]  OK     転送成功
     ERROR  転送失敗
```

解説

blockで指定したブロック中のareaで指定されるエリアを、メインメモリ上の領域のbuffに転送します。

メモリからブロックへの書き込み

機能番号 132

書式

```
STATUS  _blcwrite(block, area, buff)
HANDLE  block  [A]   ブロックハンドル
AREA    *area  [DE]  ブロック内の領域
char    *buff  [BC]  転送元アドレス
```

戻り値

```
[A]  OK     転送成功
     ERROR  転送失敗
```

解説

buffで指定したメインメモリの内容を、blockで指定したブロック中の領域areaに転送します。

ブロックへの保存

機能番号	123

書式

```
STATUS  _putblc(block, area, xoffset, yoffset)
HANDLE  block    [A']  ブロックハンドル
AREA    *area    [HL]  保存するエリア
WORD    xoffset  [BC]  ブロック内での横方向のオフセット
WORD    yoffset  [DE]  ブロック内での縦方向のオフセット
```

戻り値

[A] OK 保存成功
ERROR 保存失敗

解説

area で指定した領域を、block で参照されるブロック内の xoffset、yoffset で指定するオフセット（左上が(0, 0)、マイナスは使用できない）分右下の領域に保存します。_getblc()の逆の動作を行ないます。

ブロックから画面への表示

機能番号	124

書式

```
STATUS  _getblc(block, area, xoffset, yoffset)
HANDLE  block    [A']  ブロックハンドル
AREA    *area    [HL]  画面上のエリア
WORD    xoffset  [BC]  ブロック内の横方向のオフセット
WORD    yoffset  [DE]  ブロック内の縦方向のオフセット
```

戻り値

[A] OK 成功
ERROR 失敗

解説

area で指定したエリアに、block で参照されるブロック内の xoffset yoffset で指定するオフセット分右下の領域にある絵を、拡大縮小は行なわずに再表示します。_putblc() の逆の動作を行ないます。

ブロック内と画面の画像の交換

機能番号 125

書式

```
STATUS  _swapblc(block, area, xoffset, yoffset)
HANDLE  block    [A']  ブロックハンドル
AREA    *area    [HL]  画面上のエリア
WORD    xoffset  [BC]  ブロック内の横方向のオフセット
WORD    yoffset  [DE]  ブロック内の縦方向のオフセット
```

戻り値

```
[A]  OK     交換成功
     ERROR  交換失敗
```

解説

areaで指定した領域の画面上の画像と、blockで参照されるブロック内のxoffset. yoffsetで指定するオフセット分右下の領域にある絵を交換します。 putarea()と_getarea()とを同時に行ないます。

ブロックサイズの変更

機能番号 126

書式

```
STATUS  _resizeblc(block, xsize, ysize)
HANDLE  block  [A]   ブロックハンドル
WORD    xsize  [BC]  ブロック内の横方向のサイズ
WORD    ysize  [DE]  ブロック内の縦方向のサイズ
```

戻り値

```
[A]  OK     変更成功
     ERROR  変更失敗
```

解説

blockで参照される保存領域を、xsize、ysizeで指定した大きさに変更します。縮小時は保存されている絵の右下が切りとられ、拡大時は右下に余白がつけ加えられます。

13章 グラフパック

この章では、グラフパックの構成や使用方法、各ファンクションなどについて説明します。

13.1 グラフパックとは

グラフパックは、MSXViewの環境を維持しながらウィンドウに対して、高速に描画するためのルーチン群です。グラフパックを使えば、ペン、タイルのパターンや色などを自由に設定し、線や箱などの図形をクリッピングして描画することができます。

MSXViewでは、文字以外の画面表示はすべてグラフパックを使用します（文字の表示はフォントパックを使う）。描画することができるのは、ウィンドウ環境の描画エリア（ズームされた領域）だけです。したがって、グラフパックを使うときは、ディスプレイマネージャで描画環境を設定して下さい。描画環境の設定については、ディスプレイマネージャの章を参照して下さい。

13.2 グラフパックの使い方

グラフパックを使って描画するときは、次のような手順になります。

1. 描画に使うペンの属性を設定して、_createpen()でペンを作成し、ペンハンドルを割り付けます。ただし、すでに存在するペン（システム標準ペンSYSPENなど）を使うときは、必要ありません。
2. 描画するウィンドウを_chwin()、_pushwin()などでカレントウィンドウにします。
3. _chpen()、_pushpen()などで、使用するペンをカレントペンにします。
4. 表示を行なうエリアを_zoom()で、ズームします。
5. グラフパックの描画ファンクションを使って、実際に描画します。
6. _endzoom()で、描画状態を終了します。
7. 必要に応じて、_poppen()などで、カレントペンを元に戻します。

8 必要に応じて、_popwin() などで、カレントウィンドウを元に戻します。

13.3 描画の構成と機能

13.3.1 ペン

描画環境（線の種類、ペンの大きさ、ペンの色、塗りつぶしの色、バックの色）は PEN と呼びます。通常は、この PEN で描画されます。

13.3.2 データ構造

- 座標データ (−32768〜+32767)

```
typedef     struct      _pos {
            int         xp;
            int         yp;
}           POS;
```

- 領域データ

```
typedef     struct      _area {
            int         xp;
            int         yp;
            unsigned    xs;
            unsigned    ys;
}           AREA;
```

- カラーデータ

```
#define     COLOR       char
```

- RGB カラーデータ

```
typedef     struct      _rgb {
            TINY        red;            /* 赤の 0〜7 */
            TINY        green;          /* 緑の 0〜7 */
            TINY        blue;           /* 青の 0〜7 */
}           RGB;
```

• タイルパターンデータ

```
typedef    struct        tile {
           TINY          sw;
           COLOR         on;
           COLOR         off;
           TINY          pat[8];
}          TILE;
```

タイルスイッチ（TILE.sw ビット 4、5）
0x00　pat の中をすべて 1 と想定して、on で指定した色で書く
0x10　pat 中のビット 1 の部分のみを、off で指定した色で書く
0x20　pat 中のビット 1 の部分のみを、on で指定した色で書く
0x30　pat 中のビット 0 の部分を off で指定した色で、
　　　ビット 1 の部分を on で指定した色で書く

• ペン

```
typedef    struct        _grafpen {
           TINY          line[4];         /* 点線のパターン */
           TINY          xhot;            /* X ホットスポット */
           TINY          yhot;            /* Y ホットスポット */
           TINY          xs;              /* X サイズ */
           TINY          ys;              /* Y サイズ */
           TINY          pat[8];          /* ペンの形状 */
           TILE          pen;             /* ペンタイル */
           TILE          fill;            /* 塗りつぶしタイル */
           TILE          back;            /* バックタイル */
}          PEN;                           /*   _erasearea() など、*
                                          /*   画面をクリアするときに *
                                          /*   使用されるタイル *
```

機能番号	名前	[illegible]	ページ
195	_erasearea()	カレントペンによるエリアの塗りつぶし	[illegible]
196	_erase()	指定色でのエリアの塗りつぶし	437
197	_curtain()	エリアの塗りつぶし (OR)	437
198	_changecolor()	エリア内の指定色の変更	438
199	reverse()	エリアの塗りつぶし (XOR)	438
200	roundarea()	エリアにしたがった角の丸い四角形の描画	439
201	pixel()	カラーコードの獲得	439
202	_readbit()	エリア内のカラーコードの読み出し	439
203	_writebit()	エリアへのカラーコードの書き込み	440
297	_moveframe()	エリアの移動	436

13.5 ファンクションの説明

以下では、グラフパックの各ファンクションについて説明します。

13.5.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号　各ファンクションに割り当てられている番号です。

書　式　各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- STATUS ― ファンクションの戻り値の型
- _clearwin ― ファンクション名
- (win) ― 引数
- ウィンドウハンドル ― 引数の意味
- [A] ― 引数の受け渡しに使われる CPU レジスタ（アセンブラプログラミング時に使用）
- win ― 引数名
- HANDLE ― 引数の型

戻り値　そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- オープン成功／オープン失敗 ― 戻り値の意味
- OK／ERROR ― 戻り値
- [A] ― 戻り値の引き渡しに使われる CPU レジスタ（アセンブラプログラム時に使用）

解　説　そのファンクションがどのような動作をするかを示します。

グラフパックの初期化

機能番号　166

書式
```
void  _initgraf(void)
```

戻り値　なし

解説　グラフパックを初期化します。ペン、フォント、ウィンドウスタック PEN などを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

ペンの設定

機能番号　167

書式
```
void   setpen(pen)
PEN    *pen  [HL]  PEN構造体へのポインタ
```

戻り値　なし

解説　カレントペンを pen で指定したペンに設定し、以後この形状で描画します。

描画エリアの設定

機能番号　168

書式
```
void  _direct(area)
AREA  *area  [HL]  AREA構造体へのポインタ
```

戻り値　なし

解説　クリッピングエリア（描画領域）を直接グローバル座標で設定します。_direct() は、ウィンドウ環境を設定せずにグラフパックを使うするための特殊なファンクションです。アプリケーションでは使用しないで下さい。

図形の重なりをテストするモードの開始

機能番号 169

書 式

```
void  _testpos(xpos, ypos)
int   xpos  [BC]  テスト座標の横方向の位置
int   ypos  [DE]  テスト座標の縦方向の位置
```

戻り値 なし

解 説 このファンクションを実行してテストモードに入ると、以後すべての描画ファンクションは実際の描画を行なわず、xpos、ypos で指定した位置が図形と重なるかどうかをフラグで返します。各描画ルーチンは、重なっていたら真を返します。

エリアの重なりをテストするモードの開始

機能番号 170

書 式

```
void   testarea(area)
AREA   *area  [HL]  AREA構造体へのポインタ
```

戻り値 なし

解 説 このファンクションをコールしてテストモードに入ると、以後すべての描画ファンクションは実際の描画を行なわず、area で与えられた領域に図形が入るかを返します。すべて含まれるかどうかで、少しでもはみ出していたら入っていることにはなりません。各描画ルーチンは、領域に入っていたら偽を返します。真偽が逆なので注意して下さい。

テストモードの終了

機能番号 171

書式
```
void  _endtest(void)
```

戻り値 なし

解説 テストモードを終了します。以後、描画ファンクションは実際に描画します。

ラバーバンドモードの開始・終了

機能番号 172

書式
```
void   setrub(color)
COLOR  color  [A]  カラーコード
```

戻り値 なし

解説 指定した色が0以外のとき、ラバーバンドモードに入り、以後描画ファンクションを実行すると、図形を color で指定した色の XOR で描画します。PEN の設定は無視され、1×1のペンが使用されます。

color に0を指定して実行することで、ラバーバンドモードは終了します。

ラバーバンドカラーの獲得

機能番号 173

書式
```
COLOR  _getrub(void)
```

戻り値 [A] 色番号

解説 現在のラバーバンドのカラーコードを返します。

ペンの移動

機能番号 174

書式
```
void  _movepen(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値 なし

解説 ペンをxpos、yposで指定した位置に移動します。

点の描画

機能番号 175

書式
```
BOOL  _pset(void)
```

戻り値 [A] テストモード時は、テストの結果

解説 現在のペンの位置に点を描きます。

線の描画

機能番号 176

書式
```
BOOL  _line(xpos  ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値 [A] テストモード時は、テストの結果

解説 現在のペンの位置から指定した位置にペンを移動し、線を描きます。

四角形の描画

機能番号　177

書式
```
BOOL  _frame(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置と、xpos、yposで指定した位置を対角線とする四角形を描きます。

中を塗りつぶした四角形の描画

機能番号　178

書式
```
BOOL  _box(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置とxpos、yposで指定した位置を対角線とする、中を塗りつぶした四角形を描きます。

角の丸い四角形の描画

機能番号　179

書式
```
BOOL  _round(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置とxpos、yposで指定した位置を対角線とする、角の丸い四角形を描きます。

中を塗りつぶした角の丸い四角形の描画

機能番号	180

書式

```
BOOL  _fillround(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置とxpos、yposで指定した位置を対角線とする、中を塗りつぶした角の丸い四角形を描画します。

円の描画

機能番号	181

書式

```
BOOL  _oval(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置とxpos、yposで指定した位置を対角線とする四角形に内接する、円を描画します。

中を塗りつぶした円の描画

機能番号	182

書式

```
BOOL  _filloval(xpos, ypos)
int   xpos  [BC]  横方向のローカル座標
int   ypos  [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置とxpos、yposで指定した位置を対角線とする四角形に内接する、中を塗りつぶした円を描画します。

円弧の描画

機能番号　183

書式

```
BOOL  _arc(xpos, ypos, startangle, endangle)
int   xpos        [BC]  横方向のローカル座標
int   ypos        [DE]  縦方向のローカル座標
TINY  startangle  [A']  開始角度
TINY  endangle    [E']  終了角度
```

戻り値　[A]　テストモード時は、テストの結果

解説

startangle で指定された角度から endangle で指定された角度までの円弧を、時計回りに描画します。大きさは、現在のペンの位置と指定した xpos ypos を対角線とする四角形に内接する、円の一部となります。

扇形の描画

機能番号　184

書式

```
BOOL  _pai(xpos, ypos, startangle, endangle)
int   xpos        [BC]  横方向のサイズ指定
int   ypos        [DE]  縦方向のサイズ指定
TINY  startangle  [A']  開始角度
TINY  endangle    [E']  終了角度
```

戻り値　[A]　テストモード時は、テストの結果

解説

startangleで指定された角度から、endangle で指定された角度までの扇形を、時計回りに描画します。大きさは現在のペンの位置と指定した xpos、ypos を対角線とする四角形に内接する、円の一部となります。

中を塗りつぶした扇形の描画

機能番号　185

書　式

```
BOOL _fillpai(xpos, ypos, startangle, endangle)
int  xpos       [BC]  横方向のサイズ指定
int  ypos       [DE]  縦方向のサイズ指定
TINY startangle [A']  開始角度
TINY endangle   [E']  終了角度
```

戻り値　[A]　テストモード時は、テストの結果

解　説

startangleで指定された角度から、endangleで指定された角度までの中を塗りつぶした扇形を、時計回りに描画します。大きさは現在のペンの位置と指定したxpos yposを対角線とする四角形に内接する円の一部となります。

アイコンの描画

機能番号　186

書　式

```
void _dicon(pat, xsize, ysize)
TINY *pat   [HL]  パターンへのポインタ
WORD xsize  [BC]  アイコンの横方向のサイズ
WORD yeize  [DE]  アイコンの縦方向のサイズ
```

戻り値　なし

解　説

現在のペンの位置に、アイコン（1ビットを1ピクセルとするビットマップパターン）を描画します。

指定色によるアイコンの描画

機能番号	187

書式

```
void   _colicon(pat, xsize, ysize, oncolor, offcolor)
TINY   *pat      [HL]  アイコンパターンへのポインタ
WORD   xsize     [BC]  横方向のサイズ
WORD   ysize     [DE]  縦方向のサイズ
COLOR  oncolor   [A']  オンビット色
COLOR  offcolor  [E]   オフビット色
```

戻り値　なし

解説　現在のペンの位置に、指定された色でアイコンを描画します。

上部のみ角の丸い四角形の描画

機能番号	188

書式

```
BOOL   index(xpos, ypos)
int    xpos   [BC]  横方向のローカル座標
int    ypos   [DE]  縦方向のローカル座標
```

戻り値　[A]　テストモード時は、テストの結果

解説　現在のペンの位置と xpos、ypos で指定した位置を対角線とする、上部だけ角の丸い四角形を描画します。ペンの位置は移動しません。

多角形の描画

機能番号	189

書式

```
BOOL   _polygon(buff, num)
POS    *buff  [HL]  POS の配列へのポインタ
int    num    [DE]  buff の POS メンバの個数
```

戻り値　[A]　テストモード時は、テストの結果

解説　buff で示される POS の配列にしたがって、num 個の頂点を持つ多角形を描きます。開始点と終了点は自動的につなげられます。

中を塗りつぶした多角形の描画

機能番号 190

書式

```
BOOL  _fillpolygon(buff, num)
POS   *buff   [HL]  POSの配列へのポインタ
int   num     [DE]  buffのPOSメンバの個数
```

戻り値 [A] テストモード時は、テストの結果

解説 buffで示されるPOSの配列にしたがって、num個の頂点を持つ中を塗りつぶした多角形を描きます。開始点と終了点は自動的につなげられます。

エリア内のスクロール

機能番号 191

書式

```
void  _scroll(xsize, ysize, h, v)
WORD  xsize  [BC]   横方向のサイズ
WORD  ysize  [DE]   縦方向のサイズ
int   h      [BC']  横方向にスクロールするサイズ
int   v      [DE']  縦方向にスクロールするサイズ
```

戻り値 なし

解説 現在のペン位置を左上とするxsize、ysizeの大きさを持ったエリア内を、h、vで指定されたサイズと方向に移動し、隙間をカレントペンのバックタイルで塗りつぶします。

エリアのコピー

機能番号　192

書式

```
void _copy(xsize, ysize, dx, dy)
WORD xsize  [BC]   横方向のサイズ
WORD ysize  [DE]   縦方向のサイズ
int  dx     [BC']  コピー先の横方向の基点座標
int  dy     [DE']  コピー先の横方向の基点座標
```

戻り値　なし

解説

現在のペンの位置を左上とするxsize、ysizeの大きさを持ったエリアをdx、dyを左上とする位置にコピーします。

エリアの移動

機能番号　193

書式

```
void _move(xsize, ysize, dx, dy)
WORD xsize  [BC]   横方向のサイズ
WORD ysize  [DE]   縦方向のサイズ
int  dx     [BC']  横方向の転送先位置
int  dy     [DE']  縦方向の転送先位置
```

戻り値　なし

解説

現在のペン位置を左上とするxsize、ysizeの大きさを持ったエリアをdx、dyを左上とする位置に移動します。移動元の領域はカレントペンのバックタイル（画面をクリアするときに使用される）で塗りつぶされます。

エリアの移動

機能番号　297

書式
```
BOOL  _moveframe(area, x, y)
AREA  *area  [HL]  移動元エリア
int   x      [DE]  移動先の横方向の位置
int   y      [BC]  移動先の縦方向の位置
```

戻り値
```
[A]  TRUE   トリガボタンで終了
     FALSE  アボートボタンで終了
```

解説　エリアをポインティングデバイスの動きにしたがって移動させます。areaに最初のエリアを、xとyには最初にイベントのあった位置のグローバル座標をセットします。終了したときのイベントを_ungetevent()して戻るので、必要に応じて使うか捨てるかして下さい。

エリアにしたがった四角形の描画

機能番号　194

書式
```
void  _framearea(xsize, ysize)
WORD  xsize  [BC]  横方向のサイズ
WORD  ysize  [DE]  縦方向のサイズ
```

戻り値　なし

解説　現在のペンの位置を左上とする、xsize、ysizeの大きさを持ったエリアの外縁に沿った四角形を描画します。

カレントペンによるエリアの塗りつぶし

機能番号　195

書式

```
void   erasearea(xsize, ysize)
WORD   xsize  [BC]  横方向のサイズ
WORD   ysize  [DE]  縦方向のサイズ
```

戻り値　なし

解説　現在のペンの位置を左上とする、xsize、ysize分の大きさを持ったエリアを、カレントペンのバックタイルで塗りつぶします。

指定色でのエリアの塗りつぶし

機能番号　196

書式

```
void   _erase(xsize, ysize, color)
WORD   xsize  [BC]  横方向のサイズ
WORD   ysize  [DE]  縦方向のサイズ
COLOR  color  [A]   カラーコード
```

戻り値　なし

解説　現在のペンの位置を左上とする、xsize、ysizeの大きさを持ったエリアを、colorで指定した単色で塗りつぶします。

エリアの塗りつぶし（OR）

機能番号　197

書式

```
void   _curtain(xsize, ysize, color)
WORD   xsize  [BC]  横方向のサイズ
WORD   ysize  [DE]  縦方向のサイズ
COLOR  color  [A]   カラーコード
```

戻り値　なし

解説　現在のペンの位置を左上とする、xsize、ysizeの大きさを持ったエリアを、colorで指定した色とのORをとって塗りつぶします。

エリア内の指定色の変更

機能番号　198

書　式

```
void   _changecolor(xsize, ysize, srccolor, dstcolor)
WORD   xsize     [BC]  横方向のサイズ
WORD   ysize     [DE]  縦方向のサイズ
COLOR  srccolor  [A']  変更するカラーコード
COLOR  dstcolor  [E']  変更後のカラーコード
```

戻り値　なし

解　説

現在のペンの位置を左上とする、xsize、ysize の大きさを持ったエリア内の、srccolor で指定した色を dstcolor で指定した色に変更します。

エリアの塗りつぶし（XOR）

機能番号　199

書　式

```
void   reverse(xsize, ysize, color)
WORD   xsize  [BC]  横方向のサイズ
WORD   ysize  [DE]  縦方向のサイズ
COLOR  color  [A]   カラーコード
```

戻り値　なし

解　説

現在のペンの位置を左上とする、xsize、ysize の大きさを持ったエリアを、colorで指定した色と XOR をとって塗りつぶします。

エリアにしたがった角の丸い四角形の描画

機能番号　200

書　式
```
void   roundarea(xsize, ysize)
WORD  xsize  [BC]  横方向のサイズ
WORD  ysize  [DE]  縦方向のサイズ
```

戻り値　なし

解　説　現在のペンの位置を左上とする、xsize、ysizeの大きさを持ったエリアの外縁に沿った角の丸い四角形を描画します。

カラーコードの獲得

機能番号　201

書　式
```
COLOR  _pixel(void)
```

戻り値　[A]　カラーコード

解　説　現在のペンの位置のカラーコードを返します。

エリア内のカラーコードの読み出し

機能番号　202

書　式
```
void  _readbit(area, buff)
AREA  *area  [HL]  画面上のエリア
TINY  *buff  [DE]  読み出し先メインメモリへのポインタ
```

戻り値　なし

解　説　areaで指定されたエリアから、カラーコードをbuffに読み込みます。1ドットが1バイトのデータになります。

エリアへのカラーコードの書き込み

機能番号 203

書式
```
void _writebit(area, buff)
AREA *area [HL] AREA 構造体へのポインタ
TINY *buff [DE] 書き込み元メインメモリへのポインタ
```

戻り値 なし

解説 buff に格納されているデータを area で指定した領域に書き込みます。1ドットが1バイトのデータです。

パレットの設定

機能番号 4

書式
```
void  setpalette(color, rgb)
COLOR color [A]  カラーコード
RGB   *rgb  [DE] RGB構造体へのポインタ
```

戻り値 なし

解説 color で指定したカラーコードのパレットを、rgb の内容にしたがって変更します。

パレットの獲得

機能番号 5

書式
```
void  _getpalette(color, rgb)
COLOR color [A]  カラーコード
RGB   *rgb  [DE] パレット構造体へのポインタ
```

戻り値 なし

解説 color で指定したカラーコードののパレット値を rgb に返します。

14章 フォントパック

この章では、フォントパックの構成や各ファンクションについて説明します。

14.1 フォントパックとは

フォントパックとは、MSXViewで画面に文字を表示するためのルーチン群です。基本的には、文字コード（シフトJISコード）を与えるだけで、文字を表示できます。したがって、MS-DOSマシンとの文書データの交換が簡単にできます。また、文字表示の際には、さまざまな飾りつけ（太字、斜体、輪郭、影、縞など）を施したり、大きさを変えたりすることができます。

MSXViewでは、複数種類のフォントをディスク上に持つことができるので、ディスク上にデザインフォントを格納しておき、必要に応じて、フォントパックを使用することで、さまざまな字体の文字を画面に表示することができます。

文字を書き込むことができるのは、MSXViewウィンドウ環境の描画エリア（ズームされた領域）です。フォントパックを使用するときは、ディスプレイマネージャで描画環境を設定して下さい。描画環境の設定については、11章「ディスプレイマネージャ」を参照して下さい。

14.2 フォントパックの使い方

フォントパックで文字を表示するときは、次のような手順で処理を進めます。

1. 表示するフォントの属性を設定して、_createfont()でフォントを作成し、フォントハンドルを割り付けます。ただし、すでに存在するフォント（システム標準フォントSYSFONTなど）を使うときには、その必要はありません。
2. 描画するウィンドウを、_chwin()、_pushwin()などでカレントウィンドウにします。
3. _chfont()、_pushfont()などで、使用するフォントをカレントフォントにします。
4. 表示を行なうエリアを_zoom()でズームします。

5. _movepen() で、文字の表示位置を指定します。このとき、指定する位置は文字の左上ではなく、文字のベースラインの左端になるので、注意して下さい。
6. _dfont()、_dstr() などで、実際に文字を表示します。
7. _endzoom() で、描画状態を終了します。
8. 必要に応じて、_popfont() などで、カレントフォントを元に戻します。
9. 必要に応じて、_popwin() などで、カレントウィンドウを元に戻します。

14.3 フォントパックの構成と機能

フォントパックで表示されるフォントおよび飾りつけは、フォントテンプレートによって決定されます。また、フォントメッセージにより個々の文字の大きさなどのフォントの情報を知ることもできます。

次にフォントテンプレートについて説明します。

14.3.1 フォントテンプレート

フォントテンプレートと、それによって決定される飾りつけについて説明します。

```
/* Font template */
typedef struct  _font {
        TINY    id;             /* フォント ID */
        TINY    width;          /* サイズ  横ドット数 */
        TINY    hight;          /*         縦ドット数 */
        TINY    boldx;          /* 太字    横ドット数 */
        TINY    boldy;          /*         縦ドット数 */
        TINY    italic;         /* 斜体    32=45 度とする角度 */
        TINY    shadow;         /* 影      数 */
        COLOR   shadowc[4];     /*         色, 色, 色, 色 */
        TINY    outline;        /* 輪郭線  数 */
        COLOR   outlinec[4];    /*         色, 色, 色, 色 */
        COLOR   underline;      /* 下線    色 */
        BOOL    stripe,         /* 縞      オン・オフ */
        COLOR   fcol;           /* 文字色  色 */
        COLOR   bcol;           /* システム予約 */
        TINY    pitch;          /* 文字間  ドット */
        TINY    logic;          /* システム予約 */
        TINY    direc;          /* 文字の回転、シフト JIS コード禁止  */
}       FONT;                   /*   プロポーショナル禁止 */
```

表 3.9 フォントテンプレートの内容

名前	サイズ	意味
id	1バイト	フォントID フォントIDとは、フォントファイルの拡張子の「!」に続く文字です。その後の1文字は、フォントパターンの大きさを示します。 xxxxxxxx.!yz →フォントサイズ →フォントID →フォントファイルであることを表す →フォント名 つまり、同じフォントIDであっても、大きさの違うフォントファイルが存在します。フォントパックは、使用できるフォントファイルの内で、もっとも適した大きさのフォントパターンを拡大縮小して、表示します。フォントID「A」は、MSXView専用の漢字ROMに割り当てられています。
width	1バイト	フォントサイズ（横ドット数）
hight	1バイト	フォントサイズ（縦ドット数） フォントの縦、横の大きさを指定します。フォントのサイズは飾りつけをしないときの文字の大きさを指定します。太字などの飾りをつけると、指定したサイズより大きく表示されます。フォントパックは使用できるフォントの中から、最適なものを自動的に選び出し、拡大縮小して指定された大きさの文字を得ます。
boldx	1バイト	太字（横ドット数）
boldy	1バイト	太字（縦ドット数） 指定した縦、横のドット数分、上と右に文字を太くします。したがって、フォントパックでは、太字は必ずしも1種類ではありません。あまり大きな値を与えると、文字はつぶれます。

名前	サイズ	意味
italic	1 バイト	斜体
		指定した量だけ文字を傾けて表示します。ベースラインを基点にして、32 を 45 度として指定します。文字の大きさによって角度が変わることはありません。
shadow	1 バイト	影の数
shadowc	4 バイト	各影の色
		指定した色で影をつけて表示します。影は最大 4 枚までつけられます。影の色は 1 枚ずつ指定できるので、グラデーション風の効果を出すことができます。
outline	5 バイト	輪郭線
outlinec		文字の回りに指定した枚数、指定した色で輪郭をつけて表示します。輪郭は最大 4 重までつけられます。輪郭線の色は 1 本ずつ指定できます。
underline	1 バイト	下線
		文字の下に線を引きます。下線はベースラインの 1 ドット下に色をつけて、文字の幅で引かれます。
stripe	1 バイト	縞
		文字に縞の飾りつけをします。 0　縞の飾りなし 0 以外　縞の飾りつけ
fcol	1 バイト	文字色
		文字色を指定します。縞の飾りつけモードに指定してあるときは 1 ドットごとに指定した色が、そうでない場合はすべて指定した色が文字の色になります。
bcol	1 バイト	システム予約
pitch	1 バイト	文字間
		文字間隔をドット単位で指定します。指定したドット数が文字の幅に足され、前後の文字との間が決定します。
logic	1 バイト	システム予約

名前	サイズ	意味
direc	ビット0、1	システム予約
	ビット2	シフトJISコード禁止 ビット2がオフのときは、シフトJISコードで漢字を表示（2バイトコードの1バイト目は何も表示しない）します。ビット2がオンのときは、アスキーコード（半角平仮名を含む）で表示します。
	ビット3	プロポーショナル禁止 MSXViewでは、文字はプロポーショナルスペーシング（幅が狭い文字は、文字間隔を詰めて表示・印字を行なうこと）されるのが標準です。しかし、アプリケーションによってはこの機能が邪魔になることも考えられるので、プロポーショナルスペーシングを禁止することもできます。 0　プロポーショナルデータを持つフォントは、そのドット数だけ文字の幅が狭まる。 1　プロポーショナルスペーシングを禁止する。
	ビット4〜7	システム予約

図33フォントパックの文字

14.3.2 文字幅の計算

文字の幅は次のように計算します。

- プロポーショナル時の文字幅
  文字サイズ（テンプレートの横サイズ） + 太字 − プロポーショナル + 文字間
- プロポーショナルなしのときの文字幅
  文字サイズ
- 文字間プロポーショナルなしのときの半角文字幅
  （文字サイズ + 文字間） ÷ 2 （割り切れない場合は切捨て）

14.3.3 フォントメッセージ

フォントメッセージのデータ構造は、次のようになっています。

```
typsdsf struct _fntmsg {
        TINY    s_base;         /* 元文字のベースライン */
        TINY    s_width;        /* 元文字の横サイズ */
        TINY    s_hight;        /* 縦サイズ */

        TINY    t_base;         /* 実際のベースライン */
        TINY    t_width;        /* 実際の横サイズ */
        TINY    t_hight;        /* 縦サイズ */

        TINY    d_bass;         /* 文字のベースライン（飾りつけ後） */
        TINY    d_width;        /* 文字の最大横サイズ（飾りつけ後） */
        TINY    d_hight;        /* 縦サイズ（飾りつけ後） */
        TINY    f_pitch;        /* 表示が前にはみ出すドット数 */
        TINY    b_pitch;        /* 表示が後ろへはみ出すドット数 */
}       FNTMSG;

#define JISCOD  WORD
```

14.4 フォントファイル

14.4.1 フォーマット

フォントファイルは、次のようなフォーマットになっています。

14.4.2 フォントタイプの識別

フォント ID およびフォント名は、次のとおりファイル名で認識します。

図 3.4 フォントファイル名の意味

拡張子として使うことができるのは、以下の文字です。

表 3.10 フォントファイルの拡張子として使える文字

文字	サイズ
0 1 2 3 4 5 6 7 8 9	4~13
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	14~40

例えば、「明朝体.!F8」というフォントファイルは、次のような意味になります。

フォント名　明朝体
フォント ID　F
サイズ　12

14.4.3　フォントデータの計算

フォントデータ1文字分のバイト数の計算

フォントデータ1文字分のバイト数は、次の計算式で求めます。

　　バイト数＝（サイズ×サイズ+7)÷8

何文字入っているか

フォントファイルに文字が何文字入っているかをは、次の計算式で求めます。

　　文字数＝（ファイルサイズ−512)÷バイト数

フォントファイルを読んで、(fread) エラーが返ったら、その文字はないと判断することができます。

プロポーショナルデータ

- 1文字1バイト
  「LLLLRRRR」のように、上位4ビットで左側を、下位4ビットで右側の幅を指定します。
- データの並び

14.5 ファンクション一覧

フォントパックには、以下のファンクションがあります。

表 3.11 フォントパックのファンクション一覧

機能番号	名前	意味	ページ
133	_initfont()	フォントパックの初期化	451
134	_setfont()	フォントスタイルの変更	451
135	getfontpat()	フォントパターンの獲得	452
136	knjwidth()	全角文字の幅の獲得	452
137	_dfont()	1 文字表示	453
138	_dkanji()	全角 1 文字表示	453
139	_dpattern()	パターンの表示	453
140	_chrwidth()	文字幅の獲得	454
141	_dstr()	文字列の表示	454
142	_strwidth()	文字列の幅の獲得	454
146	_getjispat()	JIS コードによるフォントパターンの読み込み	455
147	_initfile()	フォントファイルアクセスの初期化	455
148	_readgaiji()	外字ファイルの読み込み	455

14.6 ファンクションの説明

以下では、フォントパックの各ファンクションについて説明します。

14.6.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

| 機能番号 | 各ファンクションに割り当てられている番号です。

| 書式 | 各ファンクションを使用するときの書式を示します。

| 戻り値 | そのファンクションの動作の結果、どのような値が返されるかを示します。

| 解説 | そのファンクションがどのような動作をするかを示します。

フォントパックの初期化

機能番号	133
書　式	STATUS　initfont(void)
戻り値	[A]　OK　初期化成功 ERROR　初期化失敗
解　説	フォントパックを初期化（漢字ROMのチェックやフォントスタイルの初期化など）します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。ただし、アプリケーション内で画面モードの変更をしたときは、_screen() が内部的に_initfont() をコールしているので、外字を正しく表示するために、_initfont()の後で_initffile() をコールしなければなりません。

フォントスタイルの変更

機能番号	134
書　式	FNTMSG　* setfont(font) FONT　*font　[HL]　FONT構造体へのポインタ
戻り値	[HL]　FNTMSG システムデータエリアへのポインタ
解　説	font で指定したフォントテンプレートにしたがって、フォントスタイルを設定します。ただし一般的にはフォントを変更するには、ディスプレイマネージャのフォントハンドルを使用する_createfont()、_chfont() などを使います。

フォントパターンの獲得

機能番号	135
書　式	STATUS _getfontpat(c, pat) char c [A] キャラクタコード char *pat [DE] パターンの返されるアドレス
戻り値	[A] OK 成功 ERROR 失敗
解　説	pat に c で指定した文字のフォントパターンを返します。全角文字（シフト JIS コード）を指定するときは、上位バイト、下位バイトの順で連続してコールします。上位バイトを指定したときは ERROR を返し、下位バイトを指定したときはパターンを読み込みます。

全角文字幅の獲得

機能番号	136
書　式	TINY _knjwidth(sjis) WORD sjis [HL] シフト JIS コード
戻り値	[A] 文字幅
解　説	sjis で指定したシフト JIS コードの全角文字の幅（ドット数）を返します。

1文字表示

機能番号 137

書式

```
void  _dfont(c)
char  c  [A]  キャラクタコード
```

戻り値 なし

解説 cで指定したキャラクタコードにしたがって1文字表示を行います。シフトJIS漢字コードの場合にも、上位バイト、下位バイトの順で連続してコールすると、自動的に表示されます。

全角1文字表示

機能番号 138

書式

```
void  dkanji(sjis)
WORD  sjis  〔HL〕  シフトJISコード
```

戻り値 なし

解説 シフトJISコードで文字を1文字表示します。

パターンの表示

機能番号 139

書式

```
void  _dpattern(pat)
char  *pat  〔HL〕  表示するパターンへのポインタ
```

戻り値 なし

解説 現在のフォントテンプレートにしたがった飾りつけをして、patで指定したパターンを表示します。_dpattern()は、外字やロゴ(アイコン)などに飾りつけして、表示させるためのファンクションです。ここで指定するパターンは、現在使用しているフォントと同じ大きさでなければなりません。

文字幅の獲得

機能番号　140

書　式
```
TINY _chrwidth(c)
char c [A] キャラクタコード
```

戻り値　[A] 文字幅

解　説　cで指定した文字の幅を返します。全角文字（シフト JIS コード）を指定するときは、上位バイト、下位バイトの順で連続してコールします。上位バイトを指定したときは0を返し、下位バイトを指定したときは結果を返します。

文字列の表示

機能番号　141

書　式
```
void _dstr(str)
char *str [HL] 表示する文字列（0ターミネイト）
```

戻り値　なし

解　説　strで指定した文字列を表示します。内部的には、文字列中に0があらわれるまで、繰り返し_dfont()をコールしています。

文字列幅の獲得

機能番号　142

書　式
```
WORD _strwidth(str)
char *str [HL] 文字列へのポインタ
```

戻り値　[HL] 文字列の幅

解　説　文字列の幅を返します。内部的には、文字列中に0があらわれるまで、繰り返し_chrwidth()をコールしています。

JIS コードによるフォントパターンの読み込み

機能番号　146

書式
```
STATUS  _getjispat(jis, pat)
WORD    jis   [HL]  JIS コード
char    *pat  [DE]  パターンがはいるところ
```

戻り値　[A]　OK　成功
ERROR　失敗

解説　patに、jisで指定したJISコードに相当する全角文字のフォントパターンを読み込みます。

フォントファイルアクセスの初期化

機能番号　147

書式
```
void  initffile(void)
```

戻り値　なし

解説　フォントファイルアクセスのために初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

外字ファイルの読み込み

機能番号　148

書式
```
STATUS  _readgaiji(void)
```

戻り値　[A]　OK　読み込み成功
ERROR　読み込み失敗

解説　外字ファイルをシステムデータエリアに読み込みます。

15章
テキストマネージャ

この章では、テキストマネージャの構成や各ファンクションについて説明します。

15.1 テキストマネージャとは

テキストマネージャは、MSXViewで文字列の入力や編集を行うためのマネージャです。テキストのフォーマットを指定することにより、左寄せ、右寄せ、センタリングなどを設定することができます。

テキスト編集を行うためには、以下のような情報を含むテキストテンプレートを使用します。

- テキスト編集を行うウィンドウ
  テキスト編集を行うウィンドウを指定します。
- テキスト編集領域の位置と大きさ
  テキスト編集領域をウィンドウのローカル座標で指定します。日本語入力の候補表示も、この領域内で行われます。
- テキスト編集で使用するバッファへのポインタ
  テキストマネージャを使用して文字列編集を行うためには、文字列を格納するためのバッファ（テキストバッファ）をRAM上に確保して、そこへのポインタを与えなければなりません。
- テキスト編集用のバッファの大きさ
  テキストマネージャが使用できるテキストバッファの長さを指定します。
- テキスト編集で使用するフォーマット情報
  テキストマネージャには、左寄せ、右寄せ、センタリングなどを行う機能があり、そのフォーマットを指定するだけで、自動的に表示行単位のセンタリングや右寄せが行えます（左寄せとは、通常の文字列入力）。ただし、1つのテキスト内で、複数のフォーマットを指定することはできません。

- 日本語入力を使用するかどうかを示す情報
  入力を日本語で行うかどうかを指定することができます。
- テキスト編集で使用するフォント情報
  テキスト編集で使用するフォントを指定します。ただし、日本語ワードプロセッサのように複数のフォントを1つのテキストの中に共存することはできません。
- テキスト編集可能な最大行数
  テキスト編集で行数制限を設けるときに使用します。行数制限がいらないときには、0を指定します。

以下は、アプリケーションが初期設定する必要のない情報ですが、高速化やアプリケーションでの利用のために、テキストテンプレートの中に含まれています。

- テキスト編集バッファ終端へのポインタ
- テキスト編集バッファ内のテキストの行数
- 現在表示されているテキストの先頭行
- カーソル位置へのポインタ
- カーソルの論理X、Y座標
- レンジ選択されている領域の先頭へのポインタ
- レンジ選択されている領域の終端へのポインタ

また、日本語を使うアプリケーションのために、かな漢字変換フロントエンドプロセッサインターフェイスが組み込まれ、簡単な指定を行うだけでかな漢字変換が使用できるようになっています。日本語はシフトJISコードで管理されるので、MS-DOSのテキストファイルなどと、データを交換することができます。

15.2 テキストマネージャの使い方

MSXViewアプリケーションで、文字の入力や編集をするときは、次の手順で進めます。

1. テキスト編集のためのウィンドウを作成した後、テキストバッファに編集するテキストを入れて、テキストテンプレートを設定し、_createtext()でテキストを作成し、テキストハンドルを取得します。
2. _disptext()を使用してテキストを初期化して表示します。
3. 編集対象とできるのは、全画面中で1つのテキストだけなので、どのテキストを編集対象とするかを、_chtext()、_pushtext()で指定します。_chtext()、_pushtext()をコールすると、そのテキストのカーソル位置にウィンカを用いたテキストカーソルが表示されます。

4. キーイベントを_writetext() に送るだけで、自動的にテキストが編集されます。
5. テキスト編集が終了したら、必要があれば_p●ptext() をコールして、テキスト編集環境を元に戻し、_deletetext() を用いてテキストハンドルを解放します。

テキストを編集するためには、以下のような情報を含むテキストテンプレートを作成します。

- テキスト編集を行うウィンドウハンドル
- テキスト編集領域の位置と大きさ
- テキスト編集で使用するテキストを格納するバッファへのポインタ
- テキスト編集用のバッファの大きさ
- テキスト編集で使用するフォーマット情報 (左寄せ、右寄せ、センタリング)
- 日本語入力を使用するかどうかを示す情報
- テキスト編集で使用するフォント情報
- テキスト編集可能な最大行数
- テキスト編集バッファ終端へのポインタ
- テキスト編集バッファ内のテキストの行数
- 現在表示されているテキストの先頭行
- カーソル位置へのポインタ
- カーソルの論理X、Y座標
- レンジ選択されている領域の先頭へのポインタ
- レンジ選択されている領域の終端へのポインタ

15.3 テキストマネージャの構成と機能

以下では、テキストマネージャの構成と機能について説明します。

15.3.1 テキストテンプレート

テキストマネージャは、テンプレートにしたがって動作します。テキストテンプレートのデータ構造は、次のようになっています。

```
typedef struct  _te[illegible]  {
        HANDLE  win;            /* テキスト領域の存在するウィンドウ番号 */
        AREA    area;           /* テキスト領域 (ローカル座標系) */
        char    *buff;          /* テキストバッファ先頭へのポインタ */
        int     length;         /* テキストバッファの長さ */
        WORD    opt;            /* テキスト編集における様々なオプション */
```

```
        HANDLE   font;          /* 使用するフォントハンドル */
        unsigned line;
        /******* 以下はアプリケーションが設定しなくてもよい情報 *******/
        char     *end;          /* バッファ終端へのポインタ */
        unsigned lines;         /* バッファに格納されている行数 */
        unsigned topline;       /* 表示されている先頭行 */
        char     *cursor;       /* カーソル位置へのポインタ */
        unsigned column;        /* カーソルの論理X座標 */

        unsigned row;           /* カーソルの論理Y座標 */
        char     *range;        /* 選択されている領域の先頭へのポインタ */
        char     *endrange;     /* 選択されている領域の終端へのポインタ */
}       TEXT;
```

表 3.12 テキストテンプレートの内容

名前	意味
win	テキスト編集領域の存在するウィンドウハンドルです。
area	テキスト編集領域をローカル座標で格納します。
buff	テキストバッファの先頭へのポインタです。 テキストマネージャを使用するためには、テキストを管理するためのテキストバッファをRAM上に確保しなければなりません。 テキストマネージャは、テキストバッファの先頭から編集対象となる文字列を格納します。文字列はASCII（日本語はシフトJIS）コードで格納されます。改行コードはLF（0AH）の1文字だけです（CR、LFではない）。テキストの終端は0で表します。つづけて、テキストバッファの終端から先頭に向かっての各表示行の先頭へのポインタと各表示行の幅を格納します。この2つの要素は1行につき4バイトずつ使いますが、この情報により各種の処理が格段に高速化できるため、バッファ内に保存しておくようになっています。 データの管理が、このように1つのバッファを共用する構造になっているので、アプリケーションは1つのバッファを用意するだけでテキストを処理できます。
length	テキストマネージャが使用することのできるテキストバッファの長さを示します。テキストバッファは上記のように使用されるため、バッファの長さは以下のように計算して下さい。 **length=<編集する最大文字数>+1+<編集する最大行数+2> ×4** <編集する最大文字数>は、日本語の場合1文字で2バイトとなることを考慮して決めて下さい。

名前	意味
opt	テキストマネージャがテキスト編集を行う際のさまざまなオプションを格納する領域です（デフォルトでは、0xc000が入ります）。ビットを立てると、それぞれの意味が有効になります。

ビット	内容	意味
15	ShiftJIS	シフトJISコードとして処理します。
14	kanji-Conv	かな漢字変換を有効にします。
13	Conv-Inhibit	かな漢字変換を禁止し、全角ひらがなに変換します。単語登録などの特殊用途に使用します。
12	ItemSelect	アイテムセレクタとして使用します。
11	No-Erase	表示するとき、編集領域を消しません。
10	No-Kinsoku	行頭禁則処理を行いません。
9	No-Scroll	スクロール処理を禁止します。
8	No-Zoom	ズーム処理を禁止します。一部を再表示するときに使用します。
7	No-Tabs	テキスト処理で、タブを入力しません。
6	reserved	システム予約
5	reserved	システム予約
4	WordWrap	英数字のワードラップを行います。
3	Line Space	2ビットで以下のような意味を持ちます。
2	Line Space	00:行間0行、01:行間1行、10:行間2行
1	Line Format	2ビットで以下のような意味を持ちます。
0	Line Format	00:左寄せ、01:右寄せ、10:センタリング

名前	意味
font	テキストマネージャがテキスト編集に使用するフォントハンドルです。0を格納すると、そのウィンドウのデフォルトフォントが使用されます。
line	テキストで編集可能な最大行数を指定します。0を格納すると、バッファ容量が許す限りの行を編集できます。これは、1行（または数行）の固定領域でスクロールなしのテキスト編集を行う際に使用します。
end	テキストバッファで使用されているテキスト領域の終端へのポインタです。
lines	現在テキストバッファに格納されているテキストの行数を格納します。
topline	表示エリアが小さくテキストのすべてが収まりきらないときは、テキストの一部しか画面に表示されません。この領域は、このようなときに、表示エリアの上端に表示される行の番号を格納するための領域です。
cursor	現在のカーソル位置へのポインタを格納します。

名前	意味
column	カーソルの論理X座標を格納するための領域です。カーソルが移動する度に自動的に設定されます（ただし、シフト JIS の漢字では、2 ずつ進むように数えています。つまり、行頭からのバイト数を示しているので注意して下さい）。
row	カーソルの論理 Y 座標を格納するための領域です。カーソルが移動する度に自動的に設定されます。
range	選択されている領域の先頭へのポインタを格納します。領域選択が行われていないときは、NULL を格納しています。
endrange	選択されている領域の終端へのポインタを格納します。領域選択が行われていないときは、NULL を格納しています。

15.4 テキストコントロールの使い方

テキストコントロールとは、テキストマネージャの機能を利用して、テキストをコントロールとして使用することです。テキストマネージャは、コントロール番号 15 の標準コントロールとして割り付けられており、テキストバッファの内容を表示したり、ポインティングデバイスによりカーソルの位置を指定することが、コントロールマネージャで行えます。ただし、キーボードから入力された文字の処理は、コントロールマネージャでは不可能なので、必ず_writetext() と_chtext() を使用して下さい。

テキストコントロールは、カレントテキストを自動的に切り換えるので注意して下さい。_dispcntl()、_trackcntl() などで、カレントテキストが切り換わることがあります。

15.4.1 コントロールテンプレートの形式

コントロールテンプレートは、次のような形式になっています。

```
typedef struct  _ctrltp {
        HANDLE  number;         /* コントロール番号  テキストは 15 (0FH)  */
        TINY    sw;             /* コントロールスイッチ  通常と同じ */
        int     xp              /* 有効エリア */
        int     yp              /* ここに設定しておいたエリアが、 */
        WORD    xs              /* テキストテンプレートにコピーされる。 */
        WORD    ys;
        MSG     *msg;           /* コントロールメッセージ  テキスト */
}       CONTROL;                /*   ハンドルを入れておく（ただし、 */
                                /* _createtext() しておかなければならない） */
```

コントロールを使うと、次のような利点があります。

- テキストテンプレートの作成を簡略化できます。

- テキスト編集領域の初期表示に、_dispallcntl() を利用できます。
- _trackcntl() を利用して、カーソル移動やレンジ選択を簡単に行うことができます。

コントロールマネージャの各ファンクションをコールするときの、テキストコントロールドライバ（後述）の機能を以下にまとめます。

表 3.13 テキストコントロールの機能

ファンクション名	機能
_opencntl()	_dispcntl() では、テキストの初期化処理を行うことができますが、このとき表示も行われてしまいます。_opencntl() では、内部の初期化のみを行うので、表示したくないとき（特に、_redisptext() で途中から表示したいとき）に使用します。テキストテンプレートにコントロールテンプレートの win フィールドや area フィールドをコピーする点は、_dispcntl() と同じです。
dispcntl()	_disptext() と同じことができます。さらに、カレントウィンドウを指定されるテキストテンプレートの win フィールドにコピーし、コントロールテンプレート内の有効エリアフィールドをテキストテンプレートの area フィールドにコピーします。したがって、テキストテンプレートを作成する手間が省けます。
_findpart()	常に80H を返します。これは、テキストコントロールが1つのパートから成り立つためです。
trackcntl()	コントロール内でポインティングデバイスがクリックされたときに、このルーチンを呼んでテキストコントロールを実行させます。1st ボタンが離されるまで制御は戻ってきませんが、押してすぐ離すとテキストカーソルを移動し、押してからポインティングデバイスを移動させると、領域指定（レンジ選択）を行います。
actioncntl()	テキストコントロールがレバーだけで構成されるので、何もしません。
_closecntl()	_chtext(0) と同じく、日本語入力をキャンセルします。

15.5 アイテムセレクタとして使用する方法

ここでは、テキストマネージャをアイテムセレクタとして使用する方法を説明します。

アイテムセレクタとは、いくつかのアイテム（項目）から1つのアイテムを選択するためのユーザーインターフェイスで、ファイル名の選択などに用いられます。選択するアイテムが少ないときは、単独で使いますが、アイテム数が多いときは、スクロールバーといっしょに使います。

MSXViewでは、テキストマネージャを使って、アイテムセレクタを簡単に作成することができます。

テキスト構造体をアイテムセレクタとして使うには、テキストバッファに選択すべきアイテムをLF (\n) で区切って準備しておきます (最後のアイテムには、\nは不要)。また、optフィールドのアイテムセレクタビット (ビット12) を立てておきます。

以上を準備した上で、_createtext()、_disptext()を実行すると、アイテムセレクタが表示されます。アイテムセレクタをテキストコントロールとして指定しておき、テキストコントロール内がクリックされたときに_trackcntl()を呼び出せば、ポインティングデバイスを使ってアイテムを選択することができます。選択されているアイテムは1番のウィンカ (テキストカーソルウィンカ) で示されます。ただし、レンジ指定を行うことはできません。

また、キーイベントを_writetext()に入れると、カーソル上下、ページ送り、ページ戻し、先頭、終端、行送り、行戻しなどのファンクションが機能します (挿入や削除、カーソル左右移動、レンジ選択などは行えない)。これらのキーアサインはすべてテキスト編集と共通 (キーマップによって決定される) なので、ユーザーは覚えやすくなっています。

15.6 スクロールバーのリンク方法

次に、テキストマネージャとスクロールバーのリンク方法を説明します。

テキストマネージャで管理する編集テキストの量が増えたときは、テキストにスクロールバーをつけなければなりません。MSXViewのテキストマネージャでは、以下のような簡単な方法で、スクロールバーをつけることができます。

15.6.1 スクロールバーの処理をテキストに反映させる方法

- スクロールアップアイコン (▼) をクリックしたとき
  _texteditfunc(CURSORLINEDOWN)をコールして、テキストをスクロールさせます。続けて_scrolltext()を実行しなければなりません。
- スクロールダウンアイコン (▲) をクリックしたとき
  _texteditfunc(CURSORLINEUP)をコールして、テキストをスクロールさせます。続けて_scrolltext()を実行しなければなりません。
- スクロールボックスをつかんで移動したとき
  スクロールバーテンプレート内のcurnumフィールドを使って表示先頭行を求め、_redisptext(curnum)をコールしてテキストを再表示します。

15.6.2 テキストのスクロール処理をスクロールバーに反映させる方法

writetext()、_texteditfunc()をコールしたときに、ERRORが返ったら、スクロールが必要です。_scrolltext()を呼んで、スクロールして下さい。

スクロール処理の後に、テキストテンプレートのlinesフィールド (全テキストの行数) をスクロールバーテンプレートのmaxnumフィールドに、toplineフィールド (表示されてい

る先頭行）をスクロールバーテンプレートの curnum フィールドに入れて、スクロールバーを再表示して下さい。

スクロールバーの pagenum フィールドに対応するテキストテンプレートのフィールドはないので　テキストエリアのＹサイズを、テキスト使用しているフォントの高さで割って、1 ページに表示されている行数を求めなければなりません。

15.7 ファンクション一覧

テキストマネージャには、以下のファンクションがあります。

表 3.14 テキストマネージャのファンクション一覧

機能番号	[illegible]	[illegible]	ページ
264	_inittexthd()	テキストハンドルの初期化	467
265	_createtext()	テキストの作成	467
266	_disptext()	初期化をともなうテキストの表示	468
267	_deletetext()	テキストの削除	469
268	_currenttext()	カレントテキストの獲得	469
269	_chtext()	カレントテキストの変更	470
270	_pushtext()	保存をともなったカレントテキストの変更	470
271	_poptext()	テキストの復帰	471
272	_textadrs()	テキスト構造体の獲得	471
273	_writetext()	テキストの編集	472
274	_texteditfunc()	編集ファンクションの実行	472
275	_locatetext()	テキストカーソルの移動	473
276	_setcursor()	バッファ中のテキストカーソルの移動	473
277	_scrolltext()	スクロール処理	474
278	_redisptext()	テキストの再表示	468
279	_inserttext()	文字列の挿入	474
280	_settextcursor()	テキスト位置の設定	475
284	_flushjsystem()	漢字変換のキャンセル	475

15.8 ファンクションの説明

以下では、テキストマネージャの各ファンクションについて説明します。

15.8.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号 各ファンクションに割り当てられている番号です。

書式 各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

ファ
ファ
引数
引数の意味
引数の受け渡しに使われるCPUレジスタ(アセンブラプログラミング時に使用)
引数名
引数の型

戻り値 そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

戻り値の意味
戻り値
戻り値の引き渡しに使われるCPUレジスタ(アセンブラプログラム時に使用)

解説 そのファンクションがどのような動作をするかを示します。

テキストハンドルの初期化

機能番号	264
書式	`void inittexthd(void)`
戻り値	なし
解説	テキストハンドルを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

テキストの作成

機能番号　265

書式

```
HANDLE   createtext(texttemplate, text)
TEXT     *texttemplate  [HL]  テキスト構造体へのポインタ
HANDLE   text           [E]   テキストハンドル
```

戻り値　[A]　作成されたテキストのハンドル

解説

texttemplate で示されるテキストにテキストハンドルを割り付けます。texttemplate が 0 のとき、デフォルトウィンドウ、デフォルトフォン[illegible]本語モード（かな漢字変換つき）が初期値として設定されます。[illegible]のアドレスや長さには 0 が入るので、必ず_textadrs() を使用して、[illegible]トテンプレートのアドレスを求めて、テキストバッファを設定して[illegible]text が 0 のときは新しいハンドルを割り付け、0 でないときは指定[illegible]号のハンドルが割り付けられます。

初期化をともなうテキストの表示

機能番号　266

書　式

```
STATUS  _disptext(text)
HANDLE  text  [A]  テキストハンドル
```

戻り値　[A]　表示に失敗した場合ERROR

解　説　テキストテンプレートを初期化して、テキストバッファに文字または文字列が入っているときは、それを表示します。テキスト編集を行うためには、あらかじめこのファンクションをコールしなければなりません。内部的には、このファンクションは行頭ポインタ、行の幅などを初期化します。テキストをコントロール（後述）として使うときは、_dispcntl() または_dispallcntl() でテキストを表示することができるので、_disptext() をコールする必要はありません。

注　意　表示をしない初期化を行うには、_opencntl() を使用して下さい。

テキストの再表示

機能番号　278

書　式

```
STATUS  _redisptext(line)
WORD    line  [HL]  表示開始行
```

戻り値　[A]　表示に失敗した場合ERROR

解　説　lineで指定した行を表示先頭行として、カレントテキストを再表示します。初期化はしません。

テキストの削除

機能番号	267
書式	STATUS _deletetext(text) HANDLE text [A] テキストハンドル
戻り値	[A] 失敗した場合ERROR
解説	textで指定したテキストを削除します。また、削除するテキストがカレントテキストになっている場合には、ウィンカや日本語変換候補ウィンドウも同時に消去します。

カレントテキストの獲得

機能番号	268
書式	HANDLE _currenttext(void)
戻り値	[A] テキストハンドル
解説	カレントテキストのハンドルを返します。

カレントテキストの変更

機能番号	269
書式	STATUS chtext(text) HANDLE text [A] テキストハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗

解説

カレントテキストを text で指定したテキストに変更します。このとき、テキストウィンカ（テキストカーソル）を指定したテキスト中に移動させます。ただし、ハンドルとして0を指定したときは、入力をキャンセルします。かな漢字変換が途中であれば、変換候補表示ウィンドウを閉じて変換候補を捨てます。

テキスト編集を終了させるときは、必ずこのファンクションをコールしてください。そうしないと、かな漢字変換ウィンドウが開いたままになってしまうことがあります。

注意

このファンクションでは、初期化されていないテキストをカレントテキストとすることはできません。必ず_disptext()、_dispcntl()、_opencntl()などで、テキストを初期化（行頭ポインタ、行幅テーブルの初期化）した後で、コールして下さい。

保存をともなったカレントテキストの変更

機能番号	270
書式	STATUS _pushtext(text) HANDLE text [A] テキストハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗

解説

カレントテキストをスタックに積み、text で指定したテキストをカレントとします。

テキストの復帰

機能番号	271
書式	STATUS _poptext(void)
戻り値	[A] OK 復帰成功 ERROR 復帰失敗
解説	テキストハンドルをスタックから取り出し、カレントテキストとします。_pushtext() と対にして使用します。

テキスト構造体の獲得

機能番号	272
書式	TEXT *_textadrs(text) HANDLE text [A] テキストハンドル
戻り値	[HL] テキスト構造体のアドレス
解説	text で指定されたテキストの実体である、TEXT 構造体のアドレスを返します。

テキストの編集

機能番号　273

書式

```
TINY    writetext(keyevent)
EVENT   *keyevent  [HL]  イベント構造体へのポインタ
```

戻り値　[A]　キーマップコード

解説

カレントテキストに対してキーイベントを送ります。これにより、テキスト編集が行われます。カーソルがテキスト表示エリア外に出てスクロール処理が必要になるか、ESC キーが押されるか、行数・バッファ容量などの制限により ↵ キーが挿入できなかったときに、0 以外の値（編集キーファンクションのキーマップコード）を返します。戻り値が 0 以外のときは、_scrolltext() をコールしてテキスト表示エリアをスクロールさせなければなりません。このとき必要であれば、スクロールバーを書き換えます（_scrolltext() を独立させているのはこのため）。戻り値として、キーマップコードが返るので、↵ キーや ESC キーなどが押されたことを判断することができます。これを利用して、1 行の編集操作で、RETURNKEY（keydefs.h で定義されている）が返ったら編集が終了したと判断し、ESCKEY が返ったら中断するというような処理ができます。

注意

_scrolltext() は、_writetext() から 0 以外の値が返ったときに、コールして下さい。常に_scrolltext() をコールしていると、日本語の候補表示領域にテキストカーソルウィンカが点滅するなどの問題が出ることがあります。

編集ファンクションの実行

機能番号　274

書式

```
STATUS  _texteditfunc(funccode)
TINY    funccode  [A]  編集ファンクションコード
```

戻り値　[A]　カーソルがテキスト領域外に出たら ERROR

解説

カレントテキストに対して、funccode で指定した編集ファンクションを実行します。編集ファンクションについては 608 を参照して下さい。

テキストカーソルの移動

機能番号	275
書式	STATUS　locatetext(column, row) WORD　column　[HL]　桁位置 WORD　row　[DE]　行位置
戻り値	[A]　カーソルがテキスト領域外に出たら ERROR
解説	column と row で指定した位置にテキストカーソルを移動させます。

バッファ中のテキストカーソルの移動

機能番号	276
書式	STATUS　setcursor(newcursorptr) char　*newcursorptr　[HL]　バッファ中のアドレス
戻り値	[A]　カーソルがテキスト領域外に出たら ERROR
解説	カレントテキスト中の、指定したテキストバッファへのポインタ位置にテキストカーソルを移動させます。

スクロール処理

機能番号　277

書　式

```
BOOL     _scrolltext(scrollbar)
CONTROL  *scrollbar  [HL]  スクロールバーコントロールへの
                           ポインタ
```

戻り値　[A]　TRUE　実際にスクロール処理が行なわれた
　　　　　　FALSE　スクロール処理は行なわれなかった

解　説

カレントテキストのカーソルが画面内になるように、スクロール処理を行います。このファンクションでは、縦スクロールバーを更新することができます。scrollbarに縦スクロールバーコントロールへのポインタを指定すると、スクロールが生じたときは、自動的にスクロールバーコントロールを更新し、スクロールバーを再表示します。scrollbarに0を指定したときは、スクロールバーの更新処理はしません。

文字列の挿入

機能番号　279

書　式

```
BOOL  _inserttext(ptr, text)
char  *ptr  [HL]  テキストバッファ内のアドレス
char  *str  [DE]  文字列へのポインタ
```

戻り値　[A]　TRUE　挿入が行なわれた
　　　　　　FALSE　挿入は行なわれなかった

解　説

ptrで指定したカレントテキストの位置に、strで指定した文字列を挿入します。ptrは、必ずテキストバッファ内を指し、textは0で終了するシフトJIS文字列でなければなりません。通常は、この後で、_texteditfunc(SETCURSOR)をコールして、カーソルを移動させて下さい。

テキスト位置の設定

機能番号　280

書式

```
STATUS   settextcursor(xpos, ypos)
int      xpos  [HL]  hl
int      ypos  [DE]  de
```

戻り値　[A]　設定に失敗したらERROR

解説　カレントテキストのテキストカーソルをxpos、yposで指定した位置に設定します。

漢字変換のキャンセル

機能番号　284

書式

```
void   flushjsystem(void)
```

戻り値　なし

解説　変換処理を中断し、変換中の文字列を廃棄します。

16章 リソースマネージャ

ここでは、リソースマネージャの構成や各ファンクションについて説明します。

16.1 リソースマネージャとは

リソースマネージャは、MSXViewで統一的にファイルを管理するためのマネージャです。「リソース」とは、「資源」のことです。MSXViewでは、ディスク上にファイルとして蓄えられているさまざまな情報のことを指します。

MSXViewでは、ファイルをアクセスするときは、リソースマネージャを使います。MSX-DOSプログラムに見られるような、5番地コールによるMSX-DOSの直接呼び出しは行いません。

同時に、リソースマネージャは、手間のかかるエラー処理などもサポートしています。一般的には、MSX DOSでは、アプリケーション内でのエラー処理は面倒ですが、このリソースマネージャを使うことにより、エラーメッセージの表示からリトライの実行指示まで、アプリケーションの開発にはほとんど負荷がかかりません。

また、リソースマネージャは、通常のMSX-DOSファイルシステムの呼び出し機能の他に、最大64Kバイトのファイルバッファを管理できるファイルアロケータ機能を備えています。MSXViewでは、アプリケーションプログラムのデータエリアが32Kバイトなので、大きなデータはファイルアロケータを使って、ディスク上に取らなければなりません。

また、オーバーレイプログラムの実行をサポートするための機能もリソースマネージャが管理しています。MSXViewでは、アプリケーションエリアが小さいので、大規模なアプリケーションはオーバーレイを使うことになりますが、リソースマネージャがオーバーレイに関する処理をほとんど行なうので、オーバーレイにともなった領域の待避なども含めて、多重のオーバーレイ（オーバーレイプログラムで別のオーバーレイを行なうなど）を使用することができます。

16.2 リソースマネージャの使い方

リソースマネージャは、ファイルアクセスに関するいろいろな目的で活用されるので、その使用方法はさまざまです。ここでは、リソースマネージャの使用方法を、タイプ別に分けて説明します。

16.2.1 通常のファイルをアクセスする方法

通常のファイルを読み書きするには、以下のようにします。

1. まず、ファイルをオープンします。ファイルをオープンするには、_fopen() を使用します。新規にファイルを作成するときは、_fcreate() を使うこともできます。_fopen()、_fcreate() をコールすると、ファイルハンドルが割り付けられます。以降のファイルは、このファイルハンドルでアクセスします。
2. ファイルをアクセスするには、そのファイルをカレントファイルにしなければなりま[illegible]、カレントファイルを切り換えるには、_chfile()、_pushfile() などのファンクションを使用します。
3. ファイルの読み書きには、_fread()、_fwrite() を使います。
4. ファイルアクセスが終了したら、カレントファイルを元に戻します。_pushfile() でカレントファイルを切り換えておけば、_popfile() でカレントファイルを元に戻せます。
5. _fclose() でファイルをクローズします。

16.2.2 ファイル上にデータエリアを確保して使用する方法

MSXView では、大量のデータはディスク上に確保するのが一般的です。したがって、MSXView には、ファイル上にデータエリアを確保するために、ファイル上の領域割り付けを行なうファイルアロケータが用意されています。

ファイルアロケータは、以下のような手順で使用します。

1. 作業用ファイルを_fcreate() などで作成します。また、ファイルアロケータはカレントファイルに対して有効なので、_chfile()、_pushfile() を使用してカレントファイルを切り換えておきます。
2. _initfalloc() を使用して、ファイルアロケータを初期化します。
3. 領域の割り付けが必要になったら、_falloc() で領域を割り付け、割り付けられた領域が不要になったら、_ffree() で領域を解放します。
4. 割り付けられた領域をアクセスするには、_falloc() で領域の先頭へのポインタ（ファイルのアクセスポインタなので、long 型になる）を受け取り、_fseek() でその領域をアクセスできるようにしておき、_fread()、_fwrite() でアクセスします。
5. ファイルアロケータを使用し終わったら、カレントファイルを元に戻し、ファイルをクローズします。

16.3 ファンクション一覧

リソースマネージャには、以下のファンクションがあります。

表 3.15 リソースマネージャのファンクション一覧

機能番号	[illegible]	[illegible]	ページ
217	_initres()	リソースマネージャの初期化	482
218	_setdrive()	デフォルトドライブの設定	482
219	_getdrive()	デフォルトドライブの獲得	483
220	_getfileinfo()	ファイル情報の獲得	483
221	_getdiskinfo()	ディスク情報の獲得	484
222	fset()	ファイルの検索	485
223	_fnext()	次のファイルの検索	486
224	_fopen()	ファイルのオープン	487
225	_fclose()	ファイルのクローズ	487
226	_fcreate()	ファイルの新規作成	488
227	_fdelete()	ファイルの削除	488
228	_frename()	ファイル名/ディレクトリ名の変更	489
229	_chfile()	カレントファイルの設定	489
230	pushfile()	保存をともなうカレントファイルの変更	490
231	_popfile()	カレントファイルの復帰	490
232	_fwrite()	カレントファイルへの書き込み	491
233	_fread()	カレントファイルからの読み込み	491
235	_fseek()	ファイルポインタの設定	492
236	_fpoint()	ファイルポインタの獲得	492
237	_fsize()	ファイルサイズの獲得	493
238	_ferror()	エラーコードの獲得	493
239	_msxdos()	MSX-DOS2 システムコールの実行	494
240	_initfalloc()	ファイルアロケータの初期化	494
241	_falloc()	ファイルバッファの獲得	495
242	_ffree()	ファイルバッファの解放	495
243	_absread()	論理セクタによる読み出し	496
245	_choice()	ディスクフォーマットメッセージの獲得	496
246	_format()	ディスクのフォーマット	497
247	_currentfile()	カレントファイルの獲得	497
248	_initoverlay()	オーバーレイマネージャの初期化	498
251	_execute()	オーバーレイモジュールの実行	499
252	_jump()	MSXView アプリケーションの起動	500
253	_exitmodule()	オーバーレイモジュール・チャイルドプログラムの強制終了	500

機能番号	名前	意味	ページ
254	_system()	チャイルドプログラムの起動	501
256	_modulevalue()	オーバーレイモジュール・チャイルドプログラムからの戻り値の獲得	502
300	_fmenu()	ファイルの選択	503
357	_chdir()	カレントディレクトリの変更	504
358	_getcwd()	カレントワーキングディレクトリの獲得	504
359	_mkdir()	ディレクトリの作成	505
360	_getlogin()	ディスクの接続状況の獲得	505
361	_mkfpath()	フルパス名の獲得	506
362	ffirst()	最初のエントリの検索	508
363	_fnext2()	次のエントリの検索	509
364	_dirname()	パス名の解析 (ディレクトリパス名の獲得)	510
365	basename()	パス名の解析 (ファイル名の獲得)	511
366	_fcreate2()	ファイルの新規作成(アトリビュート指定あり)	512
367	bdos()	MSX-DOS2システムコールの実行(C言語対応)	513
370	_fnew()	新しいエントリの検索	514
371	_chkversion()	MSXViewのバージョン番号の検査	515
375	_fmove()	ファイルの移動	515
376	_fgetattr()	ファイルのアトリビュートの獲得	516
377	_fsetattr()	ファイルのアトリビュートの設定	516
378	_fgetftime()	ファイルの日付と時刻の獲得	517
379	_fsettime()	ファイルの日付と時刻の設定	518
380	_fhdelete()	ファイルハンドルの削除	518
381	_fhrename()	ファイルハンドルの名前の変更	519
382	_fhmove()	ファイルハンドルの移動	519
383	_fhgetattr()	ファイルハンドルのアトリビュートの獲得	520
384	_fhsetattr()	ファイルハンドルのアトリビュートの設定	520
385	_fhgetftime()	ファイルハンドルの日付と時刻の獲得	521
386	_fhsettime()	ファイルハンドルの日付と時刻の設定	521
387	_fflush()	ディスクバッファのフラッシュ	522
393	_execute2()	オーバーレイモジュールの実行 (任意のファイル)	523
411	_cmkfpath()	ファイル作成用フルパス名の獲得	524
414	_fpathset()	複数パスからのファイルの検索の設定	525
415	_fpathnext()	複数パスからのファイルの検索	526
416	_signal()	物理エラー処理ルーチンの設定	526

16.4 ファンクションの説明

以下では、リソースマネージャの各ファンクションについて説明します。

16.4.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号 各ファンクションに割り当てられている番号です。

書式 各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- STATUS … ファンクションの戻り値の型
- _clearwin … ファンクション名
- (win) … 引数
- ウィンドウハンドル … 引数の意味
- [A] … 引数の受け渡しに使われる CPU レジスタ（アセンブラプログラミング時に使用）
- win … 引数名
- HANDLE … 引数の型

戻り値 そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- オープン失敗 … 戻り値の意味
- ERROR … 戻り値
- [A] … 戻り値の引き渡しに使われる CPU レジスタ（アセンブラプログラム時に使用）

解説 そのファンクションがどのような動作をするかを示します。

また、リソースマネージャでは、ファンクション実行時にエラーが発生すると、エラーの種類によってエラーダイアログの表示が異なります。それを、

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示の有無の説明
  物理エラー時　表示の有無の説明

という形式で説明します。

論理エラーとは、MSX–DOS2のファンクションコールが返すエラーで第2部 16.2章「ファンクションエラー」で説明されているものです。

物理エラーとは、ディスクドライバが返すエラーで、第2部 16.1章「ディスクエラー」で説明されているものです。

リソースマネージャの初期化

機能番号 217

書　式

```
void _initres(void)
```

戻り値 なし

解　説 　リソースマネージャを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

デフォルトドライブの設定

機能番号 218

書　式

```
STATUS  _setdrive(drive)
TINY    drive  [A]  ドライブ番号 (0=A:、1=B:、… 7=H:)
```

戻り値 [A]　OK　　設定成功
　　　　　ERROR　設定失敗

解　説 　デフォルトドライブを設定します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　エラーは発生しない

デフォルトドライブの獲得

機能番号	219
書　式	TINY _getdrive(void)
戻り値	[A] デフォルトドライブ (0=A:、1=B:、… 7=H:)
解　説	デフォルトドライブを返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

ファイル情報の獲得

機能番号	220

書　式

```
STATUS  _getfileinfo(fn, fcb)
char    *fn    [HL]  ファイル名
FCB     *fcb   [DE]  FCBへのポインタ
```

戻り値

```
[A]  OK     獲得成功
     ERROR  獲得失敗
```

解　説

ファイルの情報を獲得します。fnで指定したファイルを疑似的にオープンし、与えたFCB領域にファイル情報を複写することにより、ファイルのサイズや作成日付などを調べます。ファイル名にパスを含めることはできません。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ディスク情報の獲得

機能番号	221
書式	STATUS _getdiskinfo(drive, dpb) TINY drive [A] ドライブ番号 (0=A:、1=B:、… 7=H:) DPB *dpb [DE] DPBへのポインタ
戻り値	[A] OK 獲得成功 ERROR 獲得失敗
解説	ディスクの情報を獲得します。このファンクションは、MSX-DOS2のDPB (ドライブパラメータブロック) を得るためのものです。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示あり
  物理エラー時 表示あり

ファイルの検索

機能番号	222
書式	STATUS _fset(fn) char *fn [HL] ドライブ・パス・ファイル ASCIIZ 文字列
戻り値	[A] OK ファイルが見つかった ERROR ファイルが見つからなかった

解説

fn で与えられたドライブ・パス・ファイル ASCIIZ 文字列でディレクトリサーチを行ないます。ファイル名には、ワイルドカード文字として「?」を指定することができます（「*」は指定不可）。このファンクションでは、指定されたファイルが存在するかどうかを調べ、_fnext() のためにファイル名を設定します。したがって、このファンクションだけを使用しても、実際のファイル名を得ることはできません。この部分は、_ffirst() や MSX-DOS2 と動作が異なるので、注意して下さい。実際にファイル名を調べるためには、_fnext() を使用します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示あり

次のファイルの検索

機能番号

223

書式

```
STATUS  _fnext(fn)
char    *fn  [HL]  ファイル名へのポインタ
```

戻り値

[A] OK ファイルが見つかった
ERROR ファイルが見つからなかった

解説

_fset() でセットされたファイル名に基づき、実際のファイル名を次々に返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示なし
  物理エラー時 表示あり

注意

MSX-DOS2 や MSX-DOS1 の「次のエントリの検索ファンクション（_FNEXT）」とは異なり、最初のファイル名から返してきます。これは、_fset() では最初のファイル名を返さないためです。

ファイルのオープン

機能番号　224

書式
```
HANDLE  _fopen(fn, fh)
char    *fn  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
HANDLE  fh   [E]   ファイルハンドル
```

戻り値　[A]　ERROR 以外　ファイルハンドル
　　　　　　ERROR　　　オープン失敗

解説　ファイルをオープンします。fh が 0 のときに、新しいハンドルを割り付けます。fh に 0 以外の値を指定すると、そのハンドルでファイルオープンを行ないますが、特に理由がない限り、0 を入れて下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

注意　_fopen() をコールしたときは、直後にそのハンドルで、_chfile() または _pushfile() を実行しなければなりません。さもないと、他のファイルが壊れる可能性があります。

ファイルのクローズ

機能番号　225

書式
```
HANDLE  _fclose(fh)
HANDLE  fh  [A]  ファイルハンドル
```

戻り値　[A]　OK　　クローズ成功
　　　　　　ERROR　クローズ失敗

解説　ファイルをクローズします。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルの新規作成

機能番号	226

書式

```
HANDLE  _fcreate(fn, fh)
char    *fn  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
HANDLE  fh   [E]   ファイルハンドル
```

戻り値

[A] ERROR以外 ファイルハンドル
ERROR 作成失敗

解説

新しいファイルを作成します。fh が0の場合に新しいハンドルを割り付けますので、特に理由のない限り0を入れておいて下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示あり
  物理エラー時 表示あり

注意

_fcreate() をコールしたときは、直後にそのハンドルで_chfile() または_pushfile() を実行しなければなりません。さもないと、他のファイルが壊れる可能性があります。

ファイルの削除

機能番号	227

書式

```
STATUS  _fdelete(fn)
char    *fn  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
```

戻り値

[A] OK 削除成功
ERROR 削除失敗

解説

fn で指定したファイルを削除します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示あり
  物理エラー時 表示あり

ファイル名・ディレクトリ名の変更

機能番号	228

書式

```
HANDLE  _frename(fn1  fn2)
char    *fn1  [HL]  変更元のドライブ・パス・ファイル ASCIIZ
                    文字列
char    *fn2  [DE]  変更後の名前へのポインタ
```

戻り値

[A] OK 変更成功
ERROR 変更失敗

解説

fn1 で指定したファイルまたはディレクトリの名前を、fn2 で指定したものに変更します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

カレントファイルの設定

機能番号	229

書式

```
STATUS   chfile(fh)
HANDLE   fh  [A]  ファイルハンドル
```

戻り値

[A] OK 設定成功
ERROR 設定失敗

解説

カレントファイルを設定します。ファイルは、常にカレントファイルに対してアクセスされるので、カレントファイルに切り換えなければなりません。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

保存をともなうカレントファイルの変更

機能番号	230
書式	STATUS _pushfile(fh) HANDLE fh [A] ファイルハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗

解説

現在のカレントファイルをファイルスタックに積み、新しいカレントファイルを fh で指定したファイルに設定します。fh が 0 のときは、ファイルスタックに積むだけで、カレントファイルの変更は行いません。このファンクションを使うと、_popfile() をコールするだけでカレントファイルを元に戻すことができます。オーバーレイモジュールなどでファイルをアクセスするときは、このファンクションを使うと便利です。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示なし
  物理エラー時 表示なし

カレントファイルの復帰

機能番号	231
書式	STATUS _popfile(void)
戻り値	[A] OK 復帰成功 ERROR 復帰失敗

解説

ファイルスタックからファイルハンドルを戻し、カレントファイルを設定します。このファンクションは、_pushfile() と対にして使います。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示なし
  物理エラー時 表示なし

カレントファイルへの書き込み

機能番号	232

書式

```
unsigned  _fwrite(buff, n)
TINY         *buff  [HL]  バッファへのポインタ
unsigned  n         [DE]  書き込むバイト数
```

戻り値

[HL]　0001H～FFFEH　実際に書き込んだバイト数
　　　0000H　　　　ディスクフル
　　　FFFFH　　　　物理エラーが発生

解説

カレントファイルに buff からデータを n バイト書き込みます。データを書き込む位置は、_fseek() で自由に設定することができます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

カレントファイルからの読み込み

機能番号	233

書式

```
unsigned  _fread(buff, n)
TINY         *buff  [HL]  バッファへのポインタ
unsigned  n         [DE]  読み込むバイト数
```

戻り値

[HL]　0001H～FFFEH　実際に読み込んだバイト数
　　　0000H　　　　エンドオブファイル
　　　FFFFH　　　　物理エラーが発生

解説

カレントファイルからデータを n バイト読み込みます。データを読み込む位置は、_fseek() で自由に設定することができます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルポインタの設定

機能番号 235

書式

```
STATUS  fseek(point)
long    *point  [HL]  ファイルポインタへのポインタ
```

戻り値 [A] OK 設定成功
ERROR 設定失敗

解説 カレントファイルのファイルポインタを設定します。_fread()、_fwrite()で読み書きする位置がファイルポインタです。このファンクションを使えば、ファイル中の任意の位置に読み書きすることができます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

注意 ファイルポインタはlong型で、ファイルポインタへのポインタをパラメータとして渡すようになっています。

ファイルポインタの獲得

機能番号 236

書式

```
STATUS  _fpoint(point)
long    *point  [HL]  ファイルポインタへのポインタ
```

戻り値 [A] OK 獲得成功
ERROR 獲得失敗

解説 カレントファイルのファイルポインタをpointに返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルサイズの獲得

機能番号	237
書式	STATUS _fsize(length) long *length [HL] ファイルサイズへのポインタ
戻り値	[A] OK 獲得成功 ERROR 獲得失敗
解説	カレントファイルのサイズをlengthに返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

エラーコードの獲得

機能番号	238
書式	TINY _ferror(void)
戻り値	[A] 直前のエラーコード
解説	直前に起きたエラーの種類をMSX-DOS2のエラーコードで返します。MSX-DOS2のエラーコードは、2部16章「エラーおよびメッセージ」を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

MSX–DOS2システムコールの実行

機能番号	239
書　式	void _msxdos(void)　MSX-DOS2 システムコールの引数
戻り値	MSX-DOS2 システムコールの戻り値

解　説

アセンブラレベルで MSX-DOS2 のファンクションコールを実行します。各々のファンクションコールに応じた値を、CPU のレジスタに設定してコールして下さい。C 言語から MSX-DOS2 のファンクションをコールするには、_bdos() を使います。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

ファイルアロケータの初期化

機能番号	240
書　式	STATUS _initfalloc(void)
戻り値	[A]　OK　　初期化成功 　　　ERROR　初期化失敗

解　説

最大64Kバイトのファイルバッファを管理するファイルアロケータを初期化します。これは、カレントファイルに対して行われます。したがって、ファイルアロケータを使用するには、_fcreate() した後、_chfile() や_pushfile() でカレントファイルを切り換えて、_initfalloc() をコールします。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルバッファの獲得

機能番号	241
書　式	long　　*_falloc(length) unsigned　length　[HL]　ファイルバッファのサイズ
戻り値	[HL]　ファイルバッファへのポインタ

解　説

lengthで指定した長さのファイル領域を確保して、そのバッファ領域の先頭へのポインタを返します。実際にバッファを使用するときには、fseek()でバッファ領域をアクセス可能な状態にしなければなりません。戻り値が示しているバッファ領域へのポインタは、システム内のワークエリアにあり、変化する可能性があります。したがって、アプリケーションは戻り値を保存して下さい。ポインタだけを格納しておいても、後で_falloc()を実行すると、値が書き変わります。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルバッファの解放

機能番号	242
書　式	void　_ffree(freearea) long　*freearea　[HL]　ファイルバッファへのポインタ
戻り値	なし

解　説

指定したポインタで示されるバッファ領域を解放します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

論理セクタによる読み出し

機能番号 243

書式

```
STATUS    _absread(drive, sector, buff)
TINY      drive   [A]  ドライブ番号 (0=A:、1=B:、… 7=H:)
unsigned  sector  [DE] 論理セクタ番号
TINY      *buff   [BC] 読み込み用バッファへのポインタ
```

戻り値 [A] OK 読み込み成功
OK以外 読み込み失敗

解説

指定したドライブ番号、論理セクタ番号のデータをディスクから読み込みます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

ディスクフォーマットメッセージの獲得

機能番号 245

書式

```
BOOL  _choice(string, drive)
char  *string  [HL]
TINY  drive    [E]  ドライブ番号 (0=A:、1=B:、… 7=H:)
```

戻り値 [A] TRUE 獲得成功
FALSE 獲得失敗

解説

ディスクをフォーマットするための選択メッセージを、MSX-DOS2のROMから読み出します。獲得失敗が返されたときは、そのドライブはフォーマットできないドライブです。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ディスクのフォーマット

機能番号　246

書式
```
STATUS    _format(buffer, drive/type, bufflen)
TINY      *buffer      [HL]
unsigned  drive/type   [DE]
                       [D]  ドライブ番号 (0=A:、1=B:、… 7=H:)
                       [E]  タイプ
unsigned  bufflen      [BC]
```

戻り値　[A]　OK　フォーマット成功
　　　　　　ERROR　フォーマット失敗

解説　ディスクをフォーマットします。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

カレントファイルの獲得

機能番号　247

書式
```
HANDLE  _currentfile(void)
```

戻り値　[A]　ERROR 以外　ファイルハンドル
　　　　　　ERROR　獲得失敗

解説　カレントファイルハンドルを返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

オーバーレイマネージャの初期化

機能番号　248

書　式
```
void _initoverlay(void)
```

戻り値　なし

解　説

オーバーレイマネージャを初期化します。このファンクションは、アプリケーションプログラムが起動する直前に、システムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示なし

オーバーレイモジュールの実行

機能番号　251

書式

```
STATUS  _execute(module, param1, param2)
MODULE  *module  [HL]  モジュール構造体へのポインタ
char    *param1  [DE]  パラメータ1
int     param2   [BC]  パラメータ2
```

戻り値　[A]　OK　　呼び出し成功
　　　　　　ERROR　呼び出し失敗

解説

現在実行中のアプリケーションプログラムファイル内の、module.p で指定した位置から module.size の長さのオーバーレイモジュールを読み出して、実行します。param1、param2 には任意のパラメータをセットします。オーバーレイモジュールは 100H 番地からに読み込まれて実行されるため、ポインタ渡しによりデータを渡すときは、データのアドレスがオーバーレイモジュールと重ならないように注意して下さい。

オーバーレイモジュール側は

```
int main(char *param1, int param2);
```

で受けるようにします。

モジュールが終了したら、制御は呼び出し元のプログラムに戻ります。

オーバーレイモジュールが呼び出し元に何か値を返すときは、main() の戻り値またはファンクション 253 の _exitmodule() を使用します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

MSXViewアプリケーションの起動

機能番号	252

書式

```
STATUS  _jump(fn, param1, param2)
char    *fn      [HL]  ドライブ・パス・ファイルASCIIZ文字列
char    *param1  [DE]  パラメータ1
int      aram2   [BC]  パラメータ2
```

戻り値

[A] OK 起動成功
ERROR 起動失敗

解説

指定したプログラムを起動し、制御を移します。呼び出されたプログラムの実行が終了しても、呼び出し側には帰ってきません。そのモジュールが終了すると、VSHELL または環境変数 VIEWVSHELL で設定されたプログラムが起動されます。

- エラーダイアログの表示は 以下のようになります。
  論理エラー時 表示あり
  物理エラー時 表示あり

オーバーレイモジュール・子プログラムの強制終了

機能番号	253

書式

```
void      _exitmodule(ret)
unsigned  ret  [HL]  呼び出し元に渡す値
```

戻り値

なし

解説

ret で指定した戻り値を持って、オーバーレイモジュール・子プログラムを強制的に終了し、制御を呼び出し元のプログラムに移します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 エラーは発生しない
  物理エラー時 エラーは発生しない

子プログラムの起動

機能番号　254

書　式

```
STATUS  _system(fn, param1, param2)
char    *fn       [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
char    *param1   [DE]  パラメータ 1
int     param2    [BC]  パラメータ 2
```

戻り値　[A]　OK　　起動成功
　　　　　　ERROR　起動失敗

解　説

指定したプログラムを実行します。子プログラムは 100H 番地に読み込まれて実行されるので、ポインタ渡しによりデータを渡すときは、データのアドレスが子プログラムと重ならないように注意して下さい。

呼び出される子プログラムは

```
int main(char *param1, int param2);
```

で受けるようにします。

子プログラム終了したら、制御は呼び出し元のプログラムに戻ります。

子プログラムが呼び出し元に何か値を返すときは、main() の戻り値またはファンクション 253 の_exitmodule() を使用します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

オーバーレイモジュール・子プログラムからの戻り値の獲得

機能番号 256

書式 `unsigned _modulevalue(void)`

戻り値 [HL] 戻り値

解説 オーバーレイモジュール・子プログラムからの戻り値を獲得します。_execute()や_system()の戻り値は、モジュール・プログラムが起動されたかどうかなので、オーバーレイモジュール・子プログラムからの戻り値を調べるためには、このファンクションを使用します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

ファイルの選択

機能番号	300

書式

```
BOOL    _fmenu(wild, pn, ev)
char    *wild   [HL]  パスを含むワイルドカード
char    *pn     [DE]  フルパスファイル名へのポインタ
EVENT   *ev     [BC]  イベントへのポインタ
```

戻り値

[A] TRUE　選択された
　　FALSE　選択されなかった

解説

wild で指定されたワイルドカードに当てはまるファイル名を、ev で示される場所にポップアップ形式で表示します。そして、ユーザーの選択を待ち、どのファイルが選択されたかを、pnで示される場所に返します。

wild は、_mkfpath() から返された、パスを含むワイルドカードを渡します。pn は、選択されたファイル名を格納するのに充分な長さ（64 バイト以上）の領域へのポインタを渡します。ev は、このファンクションが呼ばれる原因となったイベントへのポインタを渡します。

通常は

```
if (_fmenu(_mkfpath(CAT_BIN, "????????.)??"),
        file, &event) == TRUE)
            _jump(file, param1, param2);
```

のように実行ファイルの選択に使用します。

version 1.20 以降では、wild には _mkfpath() が返す、以下の形式の文字列を与えます。

<「;」で区切られたパス並び><00H><ファイル名><00H>

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

カレントディレクトリの変更

機能番号	357

書式

```
STATUS  _chdir(dir)
char    *dir  [HL]  ドライブ・パス・ファイルASCIIZ文字列
```

戻り値

[A] OK　　変更成功
　　ERROR　変更失敗

解説

カレントディレクトリをdirで指定したディレクトリに変更します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

カレントワーキングディレクトリの獲得

機能番号	358

書式

```
STATUS  _getcwd(drive, cwd)
TINY    drive  [A]   ドライブ番号 (0=デフォルトドライブ、
                     A:=1、B:=2、… H:=8)
char    *cwd   [DE]  結果を返すメモリ領域へのポインタ
```

戻り値

[A] OK　　獲得成功
　　ERROR　獲得失敗

解説

カレントワーキングディレクトリを獲得します。cwdの示す領域は、67バイト以上必要です。返される文字列には、「<ドライブ番号>:¥」が含まれます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ディレクトリの作成

機能番号	359
書式	STATUS _mkdir(newdir) char *newdir [HL] ドライブ・パス・ファイル ASCIIZ 文字列
戻り値	[A] OK 作成成功 ERROR 作成失敗
解説	newdir で指定した名前のディレクトリを作成します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示あり
  物理エラー時 表示あり

ディスクの接続状況の獲得

機能番号	360
書式	WORD _getlogin(void)
戻り値	[HL] 論理ドライブの接続状況
解説	ディスクの接続状況を返します。ビット0にドライブA、ビット1にドライブBというように、ビットごとに各論理ドライブの接続状況が返されます。そのビットが1であれば接続されており、0であれば接続されていません。使用されるのはビット7のドライブHまでです。

- エラーダイアログの表示は 以下のようになります。
  論理エラー時 エラーは発生しない
  物理エラー時 エラーは発生しない

フルパス名の獲得

機能番号	361

書式

```
char  *_mkfpath(cat, fn)
int   cat  [HL]  カテゴリ番号
char  *fn  [DE]  ファイル名へのポインタ
```

戻り値 [HL] ドライブ・パス・ファイル ASCIIZ 文字列へのポインタ

解説

環境変数の設定に応じたフルパス名を獲得します。cat に環境変数名に対応したカテゴリ番号を、fn にパスに接続されるファイル名を指定します。フルパス名の実体は、MSXView カーネルのワークエリアにあり、他の用途と共用しています。したがって、_mkfpath の呼び出し直後に_fopen() _fcreate()、_jump()、_fmenu()、_fpathset() および_system() を呼び出す場合に限り、_mkfpath() の戻り値をそのまま渡せます。しかし、それ以外のときは、アプリケーションのワークエリアなどにコピーしてから使用して下さい。

カテゴリと環境変数の対応

カテゴリ名	環境変数名
CAT_TOP	VIEW
CAT_BIN	VIEWBIN
CAT_DA	VIEWDA
CAT_OVL	VIEWOVL
CAT_FONT	VIEWFONT
CAT_PD	VIEWPD
CAT_TEMP	TEMP
CAT_HOME	HOME
CAT_CLIP	CLIP
version 1.2 で追加されたもの	
CAT_DATA	VIEWDATA
CAT_PATH	PATH

環境変数が未定義の場合、CAT_TOP、CAT_TEMP、CAT_HOME は「A:」になります。CAT_BIN、CAT_DA、CAT_OVL、CAT_FONT、CAT_PD については、CAT_TOP を調べて、CAT_TOP が未定義ならば「A:」になります。定義されていれば、CAT_TOP の下のそれぞれBIN、DA、OVL FONT、PD となります。CAT_CLIP は CAT_HOME と同じになります。CAT_DATA はCAT_TOP と同じになります。

このファンクションは version 1.20 以降とその他で動作が異なります。

version 1.0 および 1.1 では、カテゴリ番号に相当するディレクトリ名に、指定されたファイル名が連結されるだけです。

version 1.20 以降では、カテゴリ番号に相当する環境変数に、複数のパスをセミコロン「;」で区切って記述できるようになりました。そのため、ファイル名にワイルドカードが含まれない場合は、複数のパスの中から記述された順に指定されたファイルを探し、最初に見つかったファイルの名前をフルパス名で返します。ファイルを探しているときに、そのパスのディスクドライブにフロッピーディスクが入っていないと、エラーは表示されずに、そのディスクには見つからなかったものとして、次のパスを探します。環境変数に記述されたすべてのパスを探しても見つからなかったときは、記述されたパスの最初のパスにファイル名を連結して、そのフルパス名を返します。ワイルドカードが含まれる場合は、その環境変数の値そのものにファイル名をNUL キャラクタをはさんで連結したものを返します。つまり

<環境変数の値（「;」で区切られたパス並び）><00H><ファイル名><00H>

が返されます。この戻り値はそのまま _fmenu() や _fpathset() に渡すことができます。

- エラーダイアログ表示の有無
  version 1.0 および 1.1 の場合
  論理エラー　エラーは発生しない
  物理エラー　エラーは発生しない
  version 1.20 以降の場合
  論理エラー　エラーは発生しない
  物理エラー　表示あり（drive notready を除く）

最初のエントリの検索

機能番号 362

書式

```
STATUS  _ffirst(pfib, fn, fib, attr)
char    *pfib  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
                     またはFIBポインタ
char    *fn    [DE]  検索するファイル名
FIB     *fib   [BC]  検索結果を返す FIB へのポインタ
TINY    attr   [A']  検索するファイルの属性
```

戻り値 [A] MSX-DOS2のエラーコード

解説

ファイル・ディレクトリの検索をします。

pfibに、検索するファイル・ディレクトリ(ワイルドカードを含む)もしくはFIBへのポインタを指定します。

fnには、pfibがFIBへのポインタの場合のみ、ファイル名へのポインタを指定します。pfibがファイル名へのポインタのときは、NULLを指定して下さい。

fibには、検索結果が入るFIBへのポインタを指定します。attrには、検索するファイル・ディレクトリの属性を指定します。

戻り値はMSX-DOS2のエラーコードと同じです。_fset()と違い、_ffirst()は最初のファイルを返すので注意して下さい。このファンクションは、_fset()、_fnext()とはまったく別なので、混同して使うことはできません。

ファイル属性の意味については、第2部「MSX-DOS2」のファイル情報ブロック(P.226)を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示なし
  物理エラー時 表示あり

次のエントリの検索

機能番号	363
書式	STATUS _fnext2(fib) FIB *fib [HL] 検索結果を返す FIB へのポインタ
戻り値	[A] MSX-DOS2 のエラーコード
解説	ファイル・ディレクトリの検索をします。fib には、FIB を返す領域へのポインタを指定します。戻り値は、MSX DOS2 のエラーコードと同じです。_fnext() と違い、_fnext2() は_ffirst() で見つかった次のファイルを返すので注意して下さい。このファンクションは、[illegible]、_fnext() とはまったく別なので、混同して使うことはできません。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示あり

パス名の解析（ディレクトリパス名の獲得）

機能番号　364

書式

```
STATUS  dirname(src, dst)
char    *src  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
char    *dst  [DE]  ディレクトリ名を返す領域へのポインタ
```

戻り値　[A]　MSX-DOS2 のエラーコード

解説

パス名を解析して、パス名のうち最後のファイル名以外の部分を返します。src に解析されるパス名、dst にディレクトリ名を返す領域へのポインタ（最低 64 バイトの領域が必要）を指定します。この機能は文字列操作であり、実際にそのパスが存在するかどうかは調べません。

例えば

A:¥ABC¥DEF

が渡されたときは

A:¥ABC¥

が返されます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　エラーは発生しない

パス名の解析（ファイル名の獲得）

機能番号　365

書式

```
STATUS   _basename(src, dst)
         *src  [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
char     *dst  [DE]  ファイル名を返す領域へのポインタ
```

戻り値　[A]　MSX-DDS2 のエラーコード

解説

パス名を解析して、パス名のうち最後のファイル名の部分を返します。src に解析されるパス名、dst にディレクトリ名を返す領域へのポインタ（最低 13 バイトの領域が必要）を指定します。この機能は文字列操作であり、実際にそのパスが存在するかどうかは調べません。

例えば

A:¥ABC¥DEF

が渡されたときは

DEF

が返されます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　エラーは発生しない

ファイルの新規作成（属性指定あり）

機能番号 366

書式

```
HANDLE  _fcreate2(fn, fh, attr)
char    *fn    [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
HANDLE  fh     [E]   ファイルハンドル
TINY    attr   [C]   属性
```

戻り値

[A] ERROR 以外 ファイルハンドル
ERROR 作成失敗

解説

attr で指定された属性を持つ新しいファイルを作成します。fh が 0 のときは、新しいハンドルを割り付けるので、特に理由のない限り、0 を入れて下さい。

_fcreate() との違いは、ファイル属性指定の有無だけです。

注意

_fcreate2() を使ったときは、直後にそのハンドルで _chfile() または _pushfile() を実行して下さい。さもないと、他のファイルが壊れる可能性があります。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

MSX-DOS2ファンクションの実行（C言語対応）

機能番号　367

書　式

```
void   bdos(inregs, outregs)
REGS   *inregs    [HL]  REGS構造体へのポインタ
REGS   *outregs   [HL]  REGS構造体へのポインタ
```

戻り値　なし

解　説

MSX DOS2のファンクションコールを実行します。inregsにMSX DOS2へ渡すレジスタの構造体へのポインタを、outregsにMSX DOS2から返ってきた値を格納する領域へのポインタを指定します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示あり

新しいエントリの検索

機能番号　370

書式

```
STATUS  _fnew(fn, attr, template)
char    *fn          [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
TINY    attr         [E]   ファイル属性
FIB     *template    [BC]
```

戻り値　[A]　MSX-DOS2 のエラーコード

解説

新しいファイル・ディレクトリを検索します。fn に検索するファイルまたはディレクトリ（ワイルドカードを含む）を、attr に検索するファイル・ディレクトリの属性を指定します。template に検索結果が返ります。fn および attr で指定したファイル・ディレクトリが1つでもあればその検索結果が、1つもなければファイル・ディレクトリが作成され、その結果が返ります。

詳しくは、第2部「MSX DOS2」の新しいエントリの検索（P.293）を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示なし
  物理エラー時　表示あり

MSXViewのバージョン番号の検査

機能番号	371

書式

```
STATUS    _chkversion(version)
unsigned  version  [HL]  検査するバージョン番号
```

戻り値

```
[A]    OK      適合
       ERROR   不適合
[HL]           MSXViewのバージョン番号+1
```

解説

指定したバージョン番号以上の MSXView であるかどうかを検査しバージョン番号+1を返します。

MSXView version 1.00では0x101を返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

ファイル・サブディレクトリの移動

機能番号	375

書式

```
STATUS   fmove(src, dst)
char     *src  [HL]  ファイル名またはサブディレクトリへのポインタ
char     *dst  [DE]  移動先のパス・ファイル名
```

戻り値

```
[A]  OK      移動成功
     ERROR   移動失敗
```

解説

srcで指定されたファイルまたはサブディレクトリを、dstに設定したパスに移動します。srcには移動するパス・ファイル名を、dstには移動先のパス・ファイル名を指定します。

詳しくは、第2部「MSX-DOS2」のファイル・サブディレクトリの移動(P.293)を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイル属性の獲得

機能番号 376

書式

```
STATUS  _fgetattr(fn, attr)
char    *fn    [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
TINY    *attr  [DE]  属性へのポインタ
```

戻り値

```
[A]  OK      獲得成功
     ERROR   獲得失敗
```

解説

fn で指定されたファイルの属性を attr に返します。

詳しくは、第2部「MSX-DOS2」のファイル属性の獲得・セット（P.307）を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイル属性の設定

機能番号 377

書式

```
STATUS  fsetattr(fn, attr)
char    *fn    [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
TINY    attr   [E]   ファイル属性
```

戻り値

```
[A]  OK      設定成功
     ERROR   設定失敗
```

解説

fn で指定されたファイルに、attr で指定した属性を設定します。

詳しくは、第2部「MSX-DOS2」のファイル属性の獲得・セット（P.307）を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルの日付と時刻の獲得

機能番号　378

書式
```
STATUS  _fgetftime(fn, date, time)
char    *fn     [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
int     *date   [DE]  日付へのポインタ
int     *time   [BC]  時刻へのポインタ
```

戻り値　[A]　OK　　獲得成功
　　　　　　ERROR　獲得失敗

解説

fn で指定されたファイルの日付と時刻を、それぞれ date と time で示されるメモリ領域にストアします。

詳しくは、第2部「MSX-DOS2」のファイルの日付および時刻の獲得セット (P.308) を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルの日付と時刻の設定

機能番号 379

書式

```
STATUS  _fsettime(fn, date, time)
char    *fn   [HL]  ドライブ・パス・ファイル ASCIIZ 文字列
int     date  [DE]  セットする日付の値
int     time  [BC]  セットする時刻の値
```

戻り値

```
[A]  OK     設定成功
     ERROR  設定失敗
```

解説

fn で指定されたファイルに date と timeで指定された日付と時刻をセットします。

詳しくは、第2部「MSX-DOS2」のファイルの日付および時刻の獲得・セット（P.308）を参照して下さい。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルハンドルの削除

機能番号 380

書式

```
STATUS  _fhdelete(fh)
HANDLE  fh  [A]  ファイルハンドル
```

戻り値

```
[A]  OK     削除成功
     ERROR  削除失敗
```

解説

fh で指定されたファイルを削除します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルハンドルの名前の変更

機能番号　381

書式

```
STATUS   fhrename(fh newname)
HANDLE   fh         [A]   ファイルハンドル
char     *newname   [DE]  変更する名前へのポインタ
```

戻り値　[A]　OK　変更成功
　　ERROR　変更失敗

解説　ハンドルで指定されたファイルの名前を、newnameで示される ASCIIZ 文字列の名前に変更します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルハンドルの移動

機能番号　382

書式

```
STATUS   fhmove(fh, dst)
HANDLE   fh     [A]   ファイルハンドル
char     *dst   [DE]  移動先パス名へのポインタ
```

戻り値　[A]　OK　移動成功
　　ERROR　移動失敗

解説　ハンドルで指定されたファイルを dst に設定したパスに移動します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルハンドルの属性の獲得

機能番号	383
書式	STATUS _fhgetattr(fh, attr) HANDLE fh [A] ファイルハンドル TINY *attr [DE] 属性へのポインタ
戻り値	[A] OK 獲得成功 ERROR 獲得失敗
解説	ハンドルで指定されたファイルの属性を attr に返します。 ● エラーダイアログの表示は、以下のようになります。 論理エラー時 表示あり 物理エラー時 表示あり

ファイルハンドルの属性の設定

機能番号	384
書式	STATUS _fhsetattr(hd, attr) HANDLE hd [A] ファイルハンドル TINY attr [E] 属性
戻り値	[A] OK 設定成功 ERROR 設定失敗
解説	ハンドルで指定されたファイルに、attr で指定した属性を設定します。 ● エラーダイアログの表示は、以下のようになります。 論理エラー時 表示あり 物理エラー時 表示あり

ファイルハンドルの日付と時刻の獲得

機能番号　385

書式

```
STATUS   _fhgetftime(hd, date, time)
HANDLE   hd      [A]   ファイルハンドル
int      *date   [DE]  日付へのポインタ
int      *time   [BC]  時刻へのポインタ
```

戻り値

[A]　OK　　獲得成功
　　ERROR　獲得失敗

解説

ハンドルで指定されたファイルの日付と時刻を、それぞれ date と time に返します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイルハンドルの日付と時刻の設定

機能番号　386

書式

```
STATUS   _fhsettime(hd, date, time)
HANDLE   hd      [A]   ファイルハンドル
int      date    [DE]  セットする日付の値
int      time    [BC]  セットする時刻の値
```

戻り値

[A]　OK　　設定成功
　　ERROR　設定失敗

解説

hd で指定されたファイルに、date と time で指定した日付と時刻を設定します。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ディスクバッファのフラッシュ

機能番号	387
書式	STATUS _fflush(drive) TINY drive [A] ドライブ番号 (0=デフォルトドライブ、1=A:、2=B:、… 8=H:、FFH=全ドライブ)
戻り値	[A] OK 書き込み成功 ERROR 書き込み失敗
解説	ディスクバッファ内でまだディスクに書き込んでいないデータをディスクに書き込みます。 詳しくは、第2部「MSX-DOS2」のディスクバッファのフラッシュ (P.317) を参照して下さい。 • エラーダイアログの表示は、以下のようになります。 論理エラー時 表示あり 物理エラー時 表示あり

オーバーレイモジュールの実行（任意のファイル）

機能番号　393

書式
```
STATUS  _execute2(module, param1, param2)
MODULE  *module  [HL]  モジュール構造体へのポインタ
char    *param1  [DE]  パラメータ1
int     param2   [BC]  パラメータ2
```

戻り値
[A]　OK　呼び出し成功
　　ERROR　呼び出し失敗

解説

カレントファイル内のmodule.pで指定した位置から、module.sizeの長さのオーバーレイモジュールを読み出して、実行します。param1、param2には任意のパラメータをセットします。オーバーレイモジュールは100H番地に読み込まれて実行されるため、ポインタ渡しによりデータを渡すときは、データのアドレスがオーバーレイモジュールと重ならないように注意して下さい。

オーバーレイモジュール側は

```
int main(char *param1, int param2);
```

で受けるようにします。

モジュールが終了したら、制御は呼び出し元のプログラムに戻ります。

オーバーレイモジュールが呼び出し元に何か値を返すときは、main()の戻り値またはファンクション253の_exitmodule()を使用します。

このファンクションを使用するためには、オーバーレイモジュールが入っているファイルをオープンし、_chfile()や_pushfile()でカレントファイルにしておかなければなりません。

このファンクションは、MSXViewバージョン1.10以降で使うことができます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　表示あり
  物理エラー時　表示あり

ファイル作成用フルパス名の獲得

機能番号 411

書式

```
char  *_cmkfpath(cat, fn)
int   cat  [HL]  カテゴリ番号
char  *fn  [DE]  ファイル名へのポインタ
```

戻り値 [HL] ドライブ・パス・ファイル ASCIIZ 文字列へのポインタ

解説

環境変数の設定に応じたファイル作成用のフルパス名を獲得します。cat に環境変数名に対応したカテゴリ番号を、fn にパスに接続されるファイル名を指定します。フルパス名の実体は、MSXView カーネルのワークエリアにあり、他の用途と共用しています。したがって、_mkfpath の呼び出し直後に_fopen()、_fcreate()、_jump()、_fmenu()、_fpathset() および _system() を呼び出す場合に限り、_mkfpath() の戻り値をそのまま渡せます。しかし、それ以外のときは、アプリケーションのワークエリアなどにコピーしてから使用して下さい。

version 1.20 以降では、カテゴリ番号に相当する環境変数に複数のパスを、セミコロン（「;」）で区切って記述できます。そのため、ファイルを作成するときは、_mkfpath() だけでは対応できないので、このファンクションが追加されました。このファンクションは、環境変数の値の中で最初の空でないパス名に、ファイル名を連結して返します。

_mkfpath() との使い分けは、既存のファイルを読み込むときは _mkfpath() を、ファイルを作成するときは _cmkfpath() を使用してフルパス名を作成します。この使用法をアプリケーションプログラムが守っている限り、環境変数の設定によって、ROM ディスク（FS-A1GTの機能）からファイルを読み出して、変更し、それを書き込むとき、ROM ディスクではなくフロッピーディスクやハードディスクおよび RAM ディスクに書き込むようになります。

カテゴリと環境変数の対応は _mkfpath() を参照して下さい。

このファンクションは version 1.20 以降で使用できます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

複数パスからのファイルの検索の設定

機能番号	414

書式

```
void  _fpathset(wild)
char  *wild  [HL]  パスを含むワイルドカード
```

戻り値　なし

解説

複数のパスから、適合するファイルを探すための複数パスおよびワイルドカードを含むファイル名を設定します。このファンクションは、_fpathnext() と対にして使います。_fnext() や _fnext2() と混同して使うことはできません。

wild には _mkfpath() が返す

<「;」で区切られたパス並び><00H><ファイル名><00H>

という形式で複数パスおよびファイル名を指定します。

このファンクションは version 1.20 以降で使用できます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

複数パスからのファイルの検索

機能番号 415

書式
```
STATUS  _fpathnext(pn, nthdir, fnp)
char    *pn       フルパス名
TINY    *nthdir   ディレクトリの変更回数
char    **fnp     ファイル名へのポインタ
```

戻り値 [A] MSX-DOS2 のエラーコード

解説 _fpathset() で設定された複数のパスから、適合するファイルを探します。_fset()、_fnext() と同じように、_fpathnext() が呼ばれるたびに適合するファイル名を返します。pn で示される領域に、見つかったファイルのフルパス名が返されます。fnp で示される領域に、見つかったファイル名へのポインタが返されますが、fnp が NULL であったら返されません。探している間に、次のパスに移った場合は、nthdir で示される値が +1 されますが、nthdir が NULL であったら行なわれません。

このファンクションは、_fpathset() と対にして使います。_fset() や _ffirst() と混同して使うことはできません。

このファンクションは version 1 20 以降で使用できます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時 表示なし
  物理エラー時 表示あり

物理エラー処理ルーチンの設定

機能番号 416

書式
```
TINY  (*_signal(sig, func))()
TINY  sig            [A]   エラーの種類
TINY  (*func)()      [HL]  エラー発生時の実行アドレス
```

戻り値 [HL] 以前設定されていた実行アドレス

解説 MSXView が MSX-DOS2 を呼び出して、ディスクの読み書きをしているときは、「ディスクが入っていない」や「書き込み禁止」などの物理エラーが発生することがあります。通常は物理エラーが発生したらエラーダイアログが表示されて、中止するか再実行するかがユーザーによって選

択され、その選択が MSX-DOS2 に返されます。_signal() を使用すると、物理エラーが発生したときにアプリケーションを呼び出すように設定することができます。_signal() を呼び出すと、以前設定されていたエラー処理ルーチンのアドレスを返します。

物理エラーの種類によって 3 つの処理ルーチンを別々に設定できます。1つは「書き込み禁止」、もう1つは「ディスクが入っていない」、そして、その他の物理エラーです。どのエラー処理を設定するかが sig です。

sig には以下の値を設定します。

名前	値	種類
SIGPHYERR	1	他の物理エラー
SIGWRPRTCT	2	書き込み禁止
SIGNRDY	3	ディスクが入っていない

物理エラーが発生したときは、まず「書き込み禁止」であるかどうかが調べられます。次に、「ディスクが入っていない」であるかどうかが調べられます。どちらでもない場合は、「他の物理エラー」とされます。

func はエラー処理ルーチンのアドレスです。func に SIG_DFL という値を入れて渡すとデフォルトの動作、つまり物理エラー発生時にエラーダイアログが表示されるようになります。SIG_IGN という値を入れて渡すと、物理エラー発生時には常にアボートの動作となり、エラーダイアログは表示されません。また、エラー処理ルーチンも呼ばれません。

SIG_DFL と SIG_IGN の具体的な値は以下の通りです。

名前	値
SIG_DFL	0
SIG_IGN	0FFFFH

MSXView が起動された時点では前述のとおり、SIG_DFL が設定されています。

なおアプリケーションが _signal() を使用したときは、アプリケーションの終了時点で SIG_DFL になっていなければなりません。

エラー処理ルーチンの呼ばれ方

エラー処理ルーチンは以下の関数で受けるようにします。また、エラー処理ルーチンからそのままアプリケーションの処理を続行してはなりません。つまり、必ずリターンしなければならないということです。

```
TINY  func(errcode, drive)
TINY  errcode  [A]  エラーコード
TINY  drive    [E]  ドライブ番号
```

errcodeはエラーコードで、MSX-DOS2のディスクエラーコードと同じです。具体的な値は、第2部16.1「ディスクエラー」を参照して下さい。

driveは[E]レジスタにドライブ番号(0=A:、1=B:、… 7=H:)がセットされます。

func()は、以下のどちらかを[A]で返さなければなりません。

値	意味
0	再実行する
1	アボートする

このファンクションは、MSXView version 1.2以降で使うことができます。

- エラーダイアログの表示は、以下のようになります。
  論理エラー時　エラーは発生しない
  物理エラー時　エラーは発生しない

17章 イベントマネージャ

この章では、イベントマネージャの構成や各ファンクションについて説明します。

17.1 イベントマネージャとは

イベントマネージャは、システムに対する外部からの入力を統一的に管理するマネージャです。例えば、ポインティングデバイスの動きに合わせて、カーソルを画面上で移動させる処理は、イベントマネージャが管理しています。

MSXView では、割り込み、ポインティングデバイス、キーボードが外部入力です。

17.2 イベントマネージャの使い方

ほとんどのアプリケーションはイベントを獲得して動作するので、メインループは、_getevent() を繰り返し呼ぶことになります。_getevent() で得たイベント情報に基づき、switch case 文で分岐して、各々の処理を行います。

17.3 イベントマネージャの構成と機能

以下では、イベントマネージャの構成と機能について説明します。

17.3.1 イベントの種類

イベントには、次のようなものがあります。

- トリガダウンイベント (TRIGDN)
  ポインティングデバイスの 1st ボタンが押されたときに発生します。
- トリガアップイベント (TRIGUP)
  ポインティングデバイスの 1st ボタンが離されたときに発生します。

- アボートダウンイベント (ABORT)
  ポインティングデバイスの2ndボタンが押されたときに発生します。
- アボートアップイベント (ABORTUP)
  ポインティングデバイスの2ndボタンが離されたときに発生します。
- キーボードイベント (KEYEVT)
  キーボードが押されたときに発生します。

注 意　マウスの動き (カーソルの動き) は_getevent() では知ることはできません。しかし、_getcoord() というルーチンで、ポインティングデバイスの動きを監視することはできます。

17.3.2 イベントレコード

イベント情報は、それに関するすべての情報を含むイベントレコードによって知ることができます。イベントレコードは以下のように定義されています。

```
typedef struct          _event  {
        TINY            kind;           /* イベントの種類 */
        POS             where;          /* ポインティングデバイスの位置 */
        TINY            bstat;          /* ボタンの状態 */
        char            keycode;        /* キーコード */
        TINY            kstat;          /* キーの状態 */
        TINY            keymap;         /* キーマップ */
        TINY            msg[4];         /* メッセージ */
}       EVENT;
```

表3.16 イベントレコードの内容

名前	サイズ	意味
kind	1バイト	イベントの種類 kindにはイベントの種類を識別するイベント番号が入っています。標準イベント番号は定数として、以下のように定義されています。 `#define KEYEVT 1` /* キーが押された */ `#define TRIGDN 2` /* 1stボタンが押された */ `#define TRIGUP 3` /* 1stボタンが離された */ `#define ABORT 4` /* 2ndボタンが押された */ `#define ABORTUP 5` /* 2ndボタンが離された */

<table>
<tr><th>名前</th><th>サイズ</th><th>意味</th></tr>
<tr><td>where</td><td>4バイト</td><td>ポインティングデバイスの位置
whereにはイベントが発生したときの、ポインティングデバイスの座標が入っています。座標はグローバル（画面上の絶対）座標です。
<pre>typedef struct _pos {
        WORD   xp;   /* X座標 */
        WORD   yp;   /* Y座標 */
}   POS;</pre></td></tr>
<tr><td>bstat</td><td>1バイト</td><td>ボタンの状態
bstatには、イベントが発生したときのポインティングデバイスのボタンやSHIFTキーなどの状態が入っています。それらの状態は押されているかいないかが、ビットで表現されます。ビットが立っていたら、押されていることを表します。
ビットの割り当ては以下のとおりです。
<table><tr><th>ビット</th><th>意味</th></tr><tr><td>7</td><td>システム予[illegible]</td></tr><tr><td>6</td><td>2ndボタン</td></tr><tr><td>5</td><td>1stボタン</td></tr><tr><td>4</td><td>かなキー</td></tr><tr><td>3</td><td>CAPSキー</td></tr><tr><td>2</td><td>GRAPHキー</td></tr><tr><td>1</td><td>CTRLキー</td></tr><tr><td>0</td><td>SHIFTキー</td></tr></table></td></tr>
<tr><td>keycode</td><td>1バイト</td><td>キーコード
kindがキーボードイベントのときに、このkeycodeにキーボード情報が入ります。キーボードイベントでないときは、保証し[illegible] かな キーがロックされていても、keycodeには英数が入ります。GRAPH キーまたは CTRL キーが押されているときは、すべて大文字の英数文字になります。</td></tr>
</table>

名前	サイズ	意味
kstat	1バイト	キーの状態

kstatにはキーボードイベントが発生したときのキーの種類や SHIFT キー類の状態が入ります。

ビット	意味
7	キーの種類 (0=ASCII 1=特殊)
6	常に0
5	常に0
4	かなキー
3	CAPS キー
2	GRAPH キー
1	CTRL キー
0	SHIFT キー

特殊なキーが押されると、kstatの特殊ビットが1になり 以下のコードが返ります。

キー	コード
ESC	01BH
TAB	009H
BS	008H
↵	00DH
INS	01AH
DEL	07FH
HOME	00BH
→	01CH
←	01DH
↑	01FH
↓	01EH
F1	0F1H
F2	0F2H
F3	0F3H
F4	0F4H
F5	0F5H

注 意

SELECT キーと STOP キーとは、ポインティングデバイスの 1st、2nd ボタンに割り振られるので、特殊キーコードとして得ることはできません。

名前	サイズ	意味
keymap	1バイト	キーマップ
		kindがキーボードイベントのとき、keymapにはそのとき押されたキーのハード的なキー番号が入ります。kindがキーボードイベントでないときは0が入ります。表3.17を参照して下さい。
msg	4バイト	メッセージ
		アプリケーションが自由に使用してよいエリアです。

表3.17 キーマトリックス

	キー番号（16進）							
0	0	1	2	3	4	5	6	7
8	8	9	-	^	¥	@	[	;
10	:	]	,	.	/	_	A	B
18	C	D	E	F	G	H	I	J
20	K	L	M	N	O	P	Q	R
28	S	T	U	V	W	X	Y	Z
30	SHIFT	CTRL	GRAPH	CAPS	カナ	F1	F2	F3
38	F4	F5	ESC	TAB	STOP	BS	SELECT	RETURN
40	SPACE	HOME	INS	DEL	→	↑	↓	←
	テンキー							
48	option	option	option	0	1	2	3	4
50	5	6	7	8	9	-	,	.

17.3.3 キーボード配列

イベントマネージャは、キーボードがかなロックされているときに、実際のキーボードの配列がどうなっていても、仮想的にキーボード配列を設定することができます。

キーボードの状態を表す構造体は以下のように定義されています。

```
typedef struct  _locks {
        TINY    config;         /* キーボード配列 */
        TINY    kana;           /* かなロック */
        TINY    caps;           /* CAPS ロック */
}       LOCKS;
```

表 3.18 LOCKS構造体の内容

名前	意味
config	キーボード配列
	0 ローマ字配列
	1 JIS配列
	2 五十音配列
kana	かなロック
	0 かなロックされていない
	0以外 かなロックされている
caps	CAPS ロック
	0 CAPS ロックされていない
	0以外 かなロックされている

以下で、各配列について説明します。

ローマ字配列

ローマ字配列は、QWERTY配列と同じです。ローマ字かな変換した情報がイベントになるわけではなく、QWERTY配列と同じイベントが返ります。

文字を入力するときは、テキストマネージャが現在のキーボード配列を参照して、ローマ字配列だったらローマ字かな変換を行います。この処理は、かな漢字変換の一環として扱われます。

JIS 配列

かな入力時に、キーボードを JIS 配列にします。

五十音配列

かな入力時に、キーボードを五十音配列にします。

17.3.4 ポインティングデバイス

ポインティングデバイスのサポート方法は、MSXView version 1.20以降とそれ以前では異なります。

1.20未満では、デバイスとしてキーボードとマウスに対応し、システム設定DAで設定されたポインティングデバイスに応じた処理を行ないます。

1.20 以降では、ポインティングデバイスとしてジョイスティックにも対応し、MSXView 自身かデバイスを自動判別して処理を行ないます。したがって_setdevice()、_getdevice() を積極的に使用する意味はありません。

17.3.5 カーソル

MSXView には、ポインティングカーソルおよびウィンカという 2 種類のカーソルがあります。ここでは、それぞれのカーソルについて説明します。

ポインティングカーソル

ポインティングカーソルとはいわゆるマウスカーソルのことで、ポインティングデバイスの動きに合わせて移動するカーソルとして、画面上に 1 つだけ表示されます。

表 3.19 ポインティングカーソルの内容

<table>
<tr><th>名前</th><th>意味</th></tr>
<tr><td>カーソル形状</td><td>カーソルの形状は、16×16 ドット以内の大きさなら、どのような形状にすることもできます。色は 2 色（透明を入れて 3 色）で、画面との論理演算も可能です。
カーソルパターンのテンプレート
<pre>
typedef struct _cursor {
        TINY   xhot;   /* ホットスポット */
        TINY   yhot;
        TINY   logic;   /* ロジック */
        COLOR   col1;   /* カラー   */
        TINY   pat1[32];   /* パターン 1 */
        COLOR   col2;   /* カラー   */
        TINY   pat2[32];   /* パターン 2 */
}   CURSOR;
</pre></td></tr>
<tr><td>カーソル形状番号</td><td>カーソルの形状は 12 個までで、それぞれに 1~12の番号を割り当てています。形状番号 1、2 は標準カーソルパターンに割り当てられ、3 以上はアプリケーションが定義します。
<table><tr><th>番号</th><th>名前</th><th>機能</th></tr><tr><td>1</td><td>システムカーソル</td><td>システム標準の矢印カーソル</td></tr><tr><td>2</td><td>ジョブカーソル</td><td>砂時計の形をしたカーソル
ディスクアクセスなど、処理に時間がかかるときに表示します。</td></tr></table></td></tr>
</table>

名前	意味
カーソル範囲	カーソルの移動範囲は、現在のスクリーンモードの最大範囲までです。それを変更することはできません。 スクリーンモード　X 座標　Y 座標 スクリーン 5　256　212 スクリーン 6　512　212 スクリーン 7　512　212 スクリーン 8　256　212
カーソルの移動	イベントマネージャは割り込み処理で、ポインティングデバイスの動きに合わせてカーソルを移動させます。そのときに、領域を指定しておけば、指定された領域にカーソルが入ったときだけ、カーソル形状を自動的に変えることができます。

カーソルの範囲番号	カーソル形状が自動的に変化する範囲は 16 個まで設定でき、範囲番号で表現されます。範囲番号 1 は、システムカーソルが画面いっぱいに設定します。
	カーソル範囲のテンプレート
	`typedef struct _area {`
	`        int      xp;    /* X global coordinate */`
	`        int      yp;    /* Y global coordinate */`
	`        unsigned xs;    /* Holizontal size */`
	`        unsigned ys;    /* Vertical size */`
	`}       AREA;`

カーソルレベル	カーソルレベルとは、カーソルを消すルーチン (_hide) が続けて呼び出された回数のことです。カーソルを表示するルーチン (_show) は、カーソルレベルを 1 だけ減らして 0 になったときだけ実際に表示します。したがって、hide と show のバランスをとらなければなりません。これは、何重にも hide() と show() がネストになっているときに、_show() のたびにカーソルが表示されて処理速度が遅くなるのを防ぐためです。また、ルーチンの独立性を持たせるという意味もあります。

ウィンカ

ウィンカとは、カーソルキー（↑ ↓ ← →）に追従する、明滅するカーソルのことです。ウィンカは、画面上に 1 つだけ表示されます。ウィンカは、テキストマネージャなどが使用します。

ウィンカは、_snsevent() の中で自動的に表示されます。また、_getevent() から返ったとき（イベントが発生したとき）には、ウィンカはすでに消えているので、アプリケーションは

ウィンカの存在を気にする必要はありません。ただし、_snsevent() を使用して、リアルタイム処理を含むイベント待ちを行っているときは、_chwink(0) でウィンカを消さなければなりません。

表 3.20 ウィンカの内容

<table>
<tr><th>名前</th><th>意味</th></tr>
<tr><td>ウィンカの位置と大きさの指定</td><td>ウィンカの位置と大きさの指定は、_createwinker() で行います。
形状は四角形のみです。
ウィンカ情報
<pre>
typedef struct _wink {
        HANDLE win;   /* ウィンドウ ID */
        AREA   area;   /* ウィンカの位置と大きさ */
        int   speed;   /* ウィンクスピード */
        TINY   rev;   /* リバースコード */
}   WINK;
</pre></td></tr>
<tr><td>ウィンクスピード</td><td>明と滅との間隔を等しくしたいときは、speed のビットで指定します。ビット 0 を立てると、ウィンクしません。それぞれのビットを 1 つだけ立てると、該当するスピードでウィンクします。複数のビットを立てると、明の時間が長くなります。
<table>
<tr><th>ビット</th><th>スピード</th></tr>
<tr><td>0</td><td>ウィンクなし</td></tr>
<tr><td>1</td><td>速い</td></tr>
<tr><td>⋮</td><td>⋮</td></tr>
<tr><td>15</td><td>遅い</td></tr>
</table>
明滅の間隔は、以下のアルゴリズムで行っています。したがって、speed の値によって、ある程度明滅の間隔を調節することができます。これをデューティの調節といいます。
<pre>
if ((JIFFY|1) & speed) {
        disp();
} else {
        erase();
}
</pre></td></tr>
<tr><td>リバースコード</td><td>ウィンカは、画面との XOR で表示されます。rev はそのときに使用されるコードです。通常は 0FEH を設定して下さい。</td></tr>
</table>

17.4 キーボードイベントの内部処理

イベントマネージャは、割り込み中にH_KEYI、H_TIMI、H_KEYCの3つのフックから制御を奪います。以下でそれぞれのフックについて説明します。

17.4.1 カーソル表示 (H_KEYI)

イベントマネージャはカーソル移動時のちらつきをなくすため、VDPの水平ライン割り込みを使用します。水平ライン割り込みは、毎回異なったラインに設定されます。水平ライン割り込みが発生したときは、H_KEYIで制御を奪い、カーソルを表示します。しかし、この割り込みは、RS-232Cが使用している可能性が高いので、一番先に元のフック先をコールしてから処理を行います。

17.4.2 ポインティングデバイスの読み込み (H TIMI)

この垂直帰線割り込みでは、ポインティングデバイスの読み込みを行います。処理中はVDPに対して、すべての割り込み（垂直帰線、水平ライン）を禁止させます。

17.4.3 キースキャン (H_KEYC)

MSX BIOSのキースキャンは使わず、H_KEYCで制御を奪い、キーをスキャンします。

Aレジスタに1を、HLレジスタにTABLEアドレスを入れてリターンすれば、MSXは何もしません

```
table:  defb    0
        dbfw retret
retret: ret
```

MSXViewでは、このデータをページ3に置いています。

このキースキャンは、[CAPS] キーと [かな] キーだけを処理（ランプ）し、他のキーは [SHIFT]、[GRAPH]、[かな]、[CTRL]、[CAPS] キーの状態と一緒に、キーコードをキーバッファに入れるだけです。[SELECT] キーと [STOP] キーはポインティングデバイスのボタンに対応するので、ここでは無視されます。

17.5 ファンクション一覧

イベントマネージャには、以下のファンクションがあります。

表 3.21 イベントマネージャのファンクション一覧

機能番号	名前	意味	ページ
10	_initevent()	イベントマネージャの初期化	541
12	_getevent()	イベント情報の獲得	541
13	_putevent()	イベント情報のイベントキューへの追加	541
14	_snsevent()	イベントの発生を調べる	542
15	_ungetevent()	イベント情報をキューに戻す	542
16	_getcoord()	カーソル座標の獲得	542
17	_flushevents()	イベントキューのクリア	543
18	_setkeyinfo()	キーボード状態の設定	543
19	_getkeyinfo()	キーボード状態の獲得	543
20	_setdevice()	ポインティングデバイスの設定	544
21	_initcursor()	カーソル表示の初期化	544
22	_getdevice()	ポインティングデバイスの獲得	544
25	_setpatcursor()	カーソルパターンの変更	545
26	_killpatcursor()	カーソルパターンの削除	545
27	_getpatnumber()	カレントカーソルの獲得	545
28	_setareacursor()	カーソルの有効領域の変更	546
29	_killareacursor()	カーソルの有効エリアの削除	546
30	_getareanumber()	カーソルがあるエリアの獲得	546
31	_systemcursor()	システムカーソルの設定	547
32	_jobcursor()	ジョブカーソルの設定	547
33	_initwinker()	ウィンカの初期化	547
34	_createwinker()	ウィンカの作成	548
35	_deletewinker()	ウィンカの削除	548
36	_chwinker()	カレントウィンカの変更	549
37	_pushwinker()	保存をともなうカレントウィンカの変更	549
38	_popwinker()	カレントウィンカの復帰	549
39	_currentwinker()	カレントウィンカの獲得	550
40	_winkeradrs()	ウィンカ情報の獲得	550
43	_show()	カーソルの表示	550
44	_hide()	カーソルの消去	551
45	_sync()	次の垂直同期までの待機	551

17.6 ファンクションの説明

以下では、イベントマネージャの各ファンクションについて説明します。

17.6.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号　各ファンクションに割り当てられている番号です。

書　式　各ファンクションを使用するときの書式を示します。

ファンクションの戻り値の型
ファンクション名
引数

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

引数の意味
引数の受け渡しに使われる=PUレジスタ
(アセンブラプログラミング時に使用)
引数名
引数の型

戻り値　そのファンクションの動作の結果、どのような値が返されるかを示します。

解　説　そのファンクションがどのような動作をするかを示します。

イベントマネージャの初期化

機能番号　10

書　式

```
void   initevent(void)
```

戻り値　なし

解　説　イベントマネージャを初期化します。このファンクションはシステムが自動的に実行しますので、アプリケーションから呼び出す必要はありません。

イベント情報の獲得

機能番号　12

書　式

```
EVENT   *_getevent(event)
EVENT   *event  [HL]  イベント構造体へのポインタ
```

戻り値　[HL]　イベント構造体へのポインタ

解　説　イベント情報を event に返します。このファンクションは、イベントが発生するまで制御を戻しません。アイドル状態のときに何か他の処理（時刻の表示など）をしたいときは_snsevent() を使って下さい。入力インジケータの書き換えとウィンカのブリンクも、このファンクションの中で行われます。

イベント情報のイベントキューへの追加

機能番号　13

書　式

```
void    _putevent(event)
EVENT   *event  [HL]  イベント構造体へのポインタ
```

戻り値　なし

解　説　event に設定したイベント情報をイベントキューに追加します。このファンクションはアプリケーションおよびシステムが、イベントを発生させるときに使います。

イベント発生の調査

機能番号 14

書　式
```
BOOL  _snsevent(void)
```

戻り値 [A] TRUE イベントが発生していた
FALSE イベントが発生していなかった

解　説 イベントが発生したかどうかを調べます。入力インジケータの書き換えとウィンカのブリンクも、このファンクション中で行われます。

イベント情報のイベントキューへの返還

機能番号 15

書　式
```
void   _ungetevent(event)
EVENT  *event  [HL]  イベント構造体へのポインタ
```

戻り値 なし

解　説 event で指定したイベント情報をイベントキューに戻します。

カーソル座標の獲得

機能番号 16

書　式
```
POS  *_getcoord(coord)
POS  *coord  [HL]  POS構造体へのポインタ
```

戻り値 [HL] POS構造体へのポインタ
マシン語仕様： [BC] に X、[DE] に Y 座標

解　説 現在のカーソルのグローバル座標を返します。

イベントキューのクリア

機能番号	17
書式	`void _flushevents(void)`
戻り値	なし
解説	イベントキューをクリアします。

キーボード状態の設定

機能番号	18
書式	`void _setkeyinfo(lockstat)` `LOCKS *lockstat` 〔HL〕 LOCKS構造体へのポインタ
戻り値	なし
解説	キーボード配列、かなロック、CAPS ロックの状態を設定します。

キーボード状態の獲得

機能番号	19
書式	`void _getkeyinfo(lockstat)` `LOCKS *lockstat` 〔HL〕 LOCKS構造体へのポインタ
戻り値	なし
解説	キーボード配列、かなロック、CAPS ロックの状態を lockstat に返します。

ポインティングデバイスの設定

機能番号	20

書 式

```
STATUS  _setdevice(dev)
TINY    dev  [A]  ポインティングデバイス番号
```

戻り値　[A] ERROR 設定ができなかった

解 説　ポインティングデバイスを設定します。MSXView version 1.20以降では、ポインティングデバイスを自動判別しているので、このファンクションをコールしても何もおきません。

カーソル表示の初期化

機能番号　21

書 式

```
void  _initcursor(void)
```

戻り値　なし

解 説　システムカーソルとジョブカーソルの形状を設定したあと、カーソルレベルを1に設定し、内部的に_systemcursor()を呼んでカーソルを表示します。

ポインティングデバイスの獲得

機能番号　22

書 式

```
TINY  _getdevice(void)
```

戻り値　[A] ポインティングデバイスの種類
MSXView version 1.20以降では常に1

解 説　現在設定されているポインティングデバイスの種類を返します。

カーソルパターンの変更

機能番号	25
書式	HANDLE _setpatcursor(pat, cursor) CURSOR *pat [HL] カーソルパターンへのポインタ HANDLE cursor [E] カーソルハンドル
戻り値	[A] 設定されたカーソルのハンドル
解説	cursor で指定したカーソルを pat で設定したパターンに変更します。cursor に 0 を指定したときは、新しいカーソルを設定してハンドルを返します。

カーソルパターンの削除

機能番号	26
書式	STATUS _killpatcursor(cursor) HANDLE cursor [A] カーソルハンドル
戻り値	[A] OK 削除成功 ERROR 削除失敗
解説	cursor で指定したカーソルを削除します。

カレントカーソルの獲得

機能番号	27
書式	HANDLE _getpatnumber(void)
戻り値	[A] カレントカーソルのハンドル
解説	現在表示されているカーソルのハンドルを返します。

カーソルの有効領域の変更

機能番号	28
書式	HANDLE _setareacursor(globalarea, cursor, curarea) AREA *globalarea [HL] 新たに設定する領域 HANDLE cursor [E] カーソルハンドル HANDLE curarea [C] エリアハンドル
戻り値	[A] エリアハンドル 変更に失敗したら ERROR
解説	cursor で指定したカーソルに対して、curarea で指定したエリアハンドルの有効エリアを globalarea に変更します。curarea が 0 のときは、新しいハンドルを返します。

カーソルの有効エリアの削除

機能番号	29
書式	STATUS _killareacursor(cursor) HANDLE cursor [A] カーソルハンドル
戻り値	[A] OK 削除成功 ERROR 削除失敗
解説	cursor で指定したカーソルの有効エリアを削除します。

カーソルがあるエリアの獲得

機能番号	30
書式	HANDLE _getareanumber(void)
戻り値	[A] エリアハンドル
解説	現在カーソルがある位置のエリアハンドルを返します。

システムカーソルの設定

機能番号	31
書式	HANDLE _systemcursor(void)
戻り値	[A] エリアハンドル
解説	システムカーソル（矢印カーソル）の有効エリアを画面全体に設定して、エリアハンドルを返します。

ジョブカーソルの設定

機能番号	32
書式	HANDLE _jobcursor(void)
戻り値	[A] エリアハンドル
解説	ジョブカーソル（砂時計カーソル）を画面全体に設定して、エリアハンドルを返します。

ウィンカの初期化

機能番号	33
書式	STATUS initwinker(void)
戻り値	[A] OK 初期化成功 ERROR 初期化失敗
解説	ウィンカを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

ウィンカの作成

機能番号 34

書式

```
HANDLE  _createwinker(data, winker)
WINK    *data   [HL]  WINK構造体へのポインタ
HANDLE  winker  [E]   ウィンカハンドル
```

戻り値 [A] 作成されたウィンカのハンドル

解説 winker で指定したハンドルのウィンカを作成します。winker が 0 のときは、新しいハンドルを返します。data が NULL ポインタのときは、メモリ中に構造体の領域だけが割り付けられ、ウィンカ情報はすべて 0 で埋められます。ウィンカハンドル 1 はテキストマネージャ用、ウィンカハンドル 2 はメニューマネージャ用なので、テキストを使用するときは、壊さないで下さい。

ウィンカの削除

機能番号 35

書式

```
STATUS  _deletewinker(winker)
HANDLE  winker  [A]  ウィンカハンドル
```

戻り値 [A] OK 削除成功
ERROR 削除失敗

解説 winker で指定したウィンカを削除します。

カレントウィンカの変更

機能番号	36
書式	STATUS _chwinker(winker) HANDLE winker [A] 新たにカレントとするウィンカのハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗
解説	カレントウィンカをwinkerで指定したウィンカに変更します。winkerが0のときは、画面上にあるウィンカを消去します。

保存をともなうカレントウィンカの変更

機能番号	37
書式	STATUS _pushwinker(winker) HANDLE winker [A] 新たにカレントとするウィンカのハンドル
戻り値	[A] OK 変更成功 ERROR 変更失敗
解説	カレントウィンカをウィンカスタックに積み、winkerで指定したウィンカをカレントとします。

カレントウィンカの復帰

機能番号	38
書式	STATUS _popwinker(void)
戻り値	[A] OK 変更成功 ERROR 変更失敗
解説	スタックからウィンカハンドルを取りだし、カレントウィンカとします。_pushwinker()と対にして使用します。

カレントウィンカの獲得

機能番号　39

書　式　`HANDLE _currentwinker(void)`

戻り値　[A]　カレントウィンカのハンドル

解　説　カレントウィンカのハンドルを返します。

ウィンカ情報の獲得

機能番号　40

書　式
```
WINK    *_winkeradrs(winker)
HANDLE  winker  [A]  ウィンカハンドル
```

戻り値　[HL]　WINK構造体へのポインタ

解　説　winkerで指定したウィンカのWINK構造体があるアドレスを返します。

カーソルの表示

機能番号　43

書　式　`void _show(void)`

戻り値　なし

解　説　カーソルレベルを1減らし、0になったときにだけカーソルを表示します。カーソルレベルがすでに0のときも表示され、0以下になることはありません。カーソルがすでに表示されているときは、何もしないで戻ります。

注　意　すべてのレジスタは保存されます。

カーソルの消去

機能番号　44

書　式　`void _hide(void)`

戻り値　なし

解　説　画面上からカーソルを消去し、カーソルレベルに1を加えます。すでに消されている場合は何もしません。

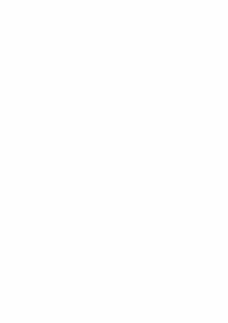
注　意　すべてのレジスタは保存されます。

次の垂直同期までの待機

機能番号　45

書　式　`void _sync(void)`

戻り値　なし

解　説　垂直同期があるまで待ちます。

18章 コントロールマネージャ

この章では、コントロールマネージャの構成や各ファンクションについて説明します。

18.1 コントロールマネージャとは

コントロールマネージャは、画面に表示されるさまざまなコントロール（ツマミ類）を管理します。コントロールとは、画面に表示されるボタン、チェックマーク、スクロールバーなどのことです。コントロールマネージャを使うことにより、これらの管理を容易にかつ効率的に行なうことができます。

コントロールを使うときは、コントロールテンプレートというデータ構造に基づいて、データを作成しておかなければなりません。コントロールテンプレートには、コントロールの種類と状態、位置（ローカル座標）、大きさなどが格納されます。このコントロールテンプレートを用意しておくと、コントロールの表示やマウスカーソルの位置を指定したコントロールの検索、実際のコントロールの動作処理などが簡単に記述できます。

また、アプリケーションで特殊なコントロー[illegible]登録することもできます。MSXViewでは、これをカスタムコントロールと呼んでいます。この場合は、コントロールドライバというモジュールを、MSXViewのルールにしたがって作成し、そのエントリを登録します。

18.2 コントロールマネージャの構成と機能

コントロールには、システムで標準的に用意されている「標準コントロール」とアプリケーションが定義する「カスタムコントロール」があります。コントロールマネージャは これらをコントロール番号を使って管理します。

18.2.1 コントロールテンプレート

コントロールは、以下のテンプレートにしたがって動作します。

```
typedef struct  _ctrltp {
        HANDLE  number;             /* コントロール番号 */
        TINY    sw;                 /* コントロールスイッチ */
        int     xp                  /* 有効エリア */
        int     yp                  /* 有効エリア */
        WORD    xs;                 /* 有効エリア */
        WORD    ys;                 /* 有効エリア */
        MSG     *msg;               /* コントロールメッセージ */
}       CONTROL;
```

表 3.22 コントロールテンプレートの内容

名前	意味
コントロール番号	コントロール番号とは、コントロールの種類を指す番号のことで、標準とカスタムがあります。標準コントロール用は 0~63 までで、64~255 はカスタムコントロール用です。
コントロールスイッチ	コントロールの状態を設定するスイッチで、ビット単位で指定します。

ビット	名前	意味
7	ハイライト	強調表示
6	マスク	表示されない
5	ディスエーブル	_findcntl() で検索されない。
4	システム予約	
3	システム予約	
2	システム予約	
1	システム予約	
0	エンド	最後の項目であることを示す

名前	意味
有効エリア	そのコントロールの有効エリアが入ります。カーソルの位置がそのコントロールの上にあるかどうか調べる _testcntl() や、どのコントロールの上にあるかを調べる _findcntl() などで使用します。
コントロールメッセージ	各コントロールのテンプレートアドレスなどが入ります。データの型およびテンプレートの内容は、各コントロールによって異なるので、(MSG *) でキャストして使用して下さい。

18.2.2　パート番号

パート番号とは、1つのコントロールが複数の部分に分かれているときに、それぞれの部分を区別・認識するために使われる番号のことです。_findpart() ファンクションを使用することで、カーソルの位置がどのパートの上にあるか知ることができます。

パート番号は1から始めます。レバー（つかんで動かすことのできるパートのことをレバーと呼ぶ）のパート番号は128以上です。

18.2.3　標準コントロールの機能と使用方法

標準コントロールには、以下のようなものがあります。

表 3.23 標準コントロールの種類

名前	意味
NULL	何もしません
ボタン 設定	中に文字が入った角の丸い長方形
チェックマーク ✓	それが選択されていることを示すマーク
コントロールボタン ◉	ON、OFF スイッチ
横スクロールバー	横方向にスクロールさせるツマミ
縦スクロールバー	縦方向にスクロールさせるツマミ
アイコン	アイコン
フレーム	四角
ライン	直線
ラウンド	角の丸い四角
イレース	領域の塗りつぶし
文字列	文字列の表示
テキスト編集	テキストの編集

これら標準コントロールはメッセージに各々のテンプレートアドレスを入れて_opencntl()や_trackcntl()で使用します。

各々のテンプレートや構成は、18.3「標準コントロールの説明」を参照して下さい。

18.2.4 カスタムコントロール

アプリケーションは標準コントロールのほかに、独自のコントロールを加えることができます。MSXViewでは、このアプリケーション独自のコントロールのことを「カスタムコントロール」と呼びます。

3方式選択スイッチ、温度計、ダイアル、カラーメニューなどアプリケーションは必要に応じてコントロールドライバ（後述）を作成し、それを定義しておけば標準コントロールと同様に使用することができます。

注　意

カスタムコントロールドライバはアプリケーションエリアに位置するのでオーバーレイした場合にそのコントロールを使用すると暴走する恐れがあります。

オーバーレイ先でそのコントロールを使用したい場合はコントロールドライバのアドレスをアプリケーションエリアの上位アドレス（例えば7100H以上）に置いてオーバーレイモジュールはそれを壊さない大きさ（例えば、7000H以下）にします。

18.3 標準コントロールの説明

ここでは、標準コントロールの内容について説明します。

18.3.1 標準コントロールの定義

ウィンドウのデフォルトPENおよびデフォルトFONTを使用するコントロールは以下のとおりです。

```
#define NULL_CNTL        1     /* NULL */
#define BUTTON_CNTL      2     /* ボタン */
#define MARK_CNTL        3     /* チェックマーク */
#define CNTL_CNTL        4     /* コントロールボタン */
#define HBAR_CNTL        5     /* 横スクロールバー */
#define VBAR_CNTL        6     /* 縦スクロールバー */
#define CICON_CNTL       7     /* 色付きアイコン */
#define ICON_CNTL        8     /* アイコン */
#define FRAME_CNTL       9     /* フレーム */
#define LINE_CNTL        10    /* 線 */
#define ROUND_CNTL       11    /* 角の丸い四角 */
#define ERASE_CNTL       12    /* 1色で塗りつぶす */
#define STRING_CNTL      13    /* 文字列 */
#define TEXT_CNTL        15    /* テキスト */
```

カレント PEN およびカレント FONT を使用するコントロールは以下のとおりです。

```
#define BUTTON_STD      17
#define MARK_STD        18
#define CNTL_STD        19
#define HBAR_STD        20
#define VBAR_STD        21
#define LINE_STD        24
#define ROUND_STD       25
#define STRING_STD      26
```

18.3.2 標準コントロールの内容

ここでは、標準コントロールの内容について説明します。

表記法

標準コントロールの内容については、以下の表記で説明します。

標準コントロールの名前

書　式	_opencntl() や_dispcntl() などのファンクションに渡す、コントロール構造体の書式を示します。
表　示	_dispcntl() などがコールされたときに、どのように表示されるかを示します。
動　作	このコントロールの上で、クリックまたはドラッグされたときに、どういう動作を行うかを示します。

NULL (NULL_CNTL)

書　式
```
static  CONTROL   cntl[ ] = { NULL_CNTL, sw, xp, yp, xs,
                              ys, (MSG *)0 };
```

表　示　なし

動　作　なし
コントロールドライバを使い、領域のチェックを行うために使用されます。

ボタン (BUTTON_CNTL)

書　式
```
static  CONTROL   cntl[ ] = { BUTTON_CNTL, sw, xp, yp,
                              xs, ys, "文字列" };
```

表　示　角の丸い四角の中に、文字列がセンタリングで表示されます。

設定

動　作　ドラッグ中は、リバースします。

チェックマーク (MARK_CNTL)

書　式
```
static  TINY      flag;
static  CONTROL   cntl[ ] = { MARK_CNTL, sw, xp, yp, xs,
                              ys, (MSG *)&flag };
```

表　示　flag のオン・オフによって、チェックマークが表示されます。横方向は指定した領域の左端に、縦方向は中央に表示されます。

√

動　作　ドラッグ中は、リバースします。

コントロールボタン（CNTL_CNTL）

書式

```
static  TINY      flag
static  CONTROL   cntl[ ] = { CNTL_CNTL, sw, xp, yp, xs
                  ys, (MSG *)&flag };
```

表示

flagのオン・オフによって、コントロールマークが表示されます。横方向は指定した領域の左端に、縦方向は中央に表示されます。

◉

動作

ドラッグ中は、リバースします。

横スクロールバー（HBAR_CNTL）

書式

```
static  BARTMP    scrollh = { 0, 100, 20 };
static  CONTROL   cntl[ ] = { HBAR_CNTL, sw, xp, yp, xs,
                  ys, (MSG *)&scrollh };
```

表示

領域を表すフレームと、領域のどのあたりを表示しているかを表すスクロールボックスとを表示します。

矢印はアイコンを使って表示

動作

横方向に長いオブジェクトの一部が画面に表示されているとき、そのオブジェクトの位置を移動するために使用します。スクロールバーのどこをクリックされたのかは、_findpart()で返されるパート番号で知ることができます。これを利用して、実際のオブジェクトの移動を行います。また　ボックスを直接ドラッグしたときは、スクロールバーテンプレートを参照することにより、指定した位置を知ることができます。ただし、_trackcntl()では、スクロールバーは描き直さないので、アプリケーションが描き直さなければなりません。

ダイアログマネージャを使うと、COMMAND.funcに以下の値が返ります。

PAGEL　なし（ボックスの左）
PAGER　なし（ボックスの右）
MOVEH　ドラッグ中、スクロールボックスが動きます。

縦スクロールバー (VBAR_CNTL)

書 式

```
static  BARTMP    scrollv = { 0, 100, 20 };
static  CONTROL   cntl[ ] = { VBAR_CNTL, sw, xp, yp, xs,
                  ys, (MSG *)&scrollv };
```

表 示

領域を表すフレームと、領域のどのあたりを表示しているかを表すスクロールボックスとを表示します。

矢印はアイコンを使って表示

動 作

縦方向に長いオブジェクトの一部が画面に表示されているとき、そのオブジェクトの位置を移動するために使用します。スクロールバーのどこをクリックされたのかは、_findpart() で返されるパート番号で知ることができます。これを利用してオブジェクトを移動します。また、ボックスを直接ドラッグしたときは、スクロールバーテンプレートを参照することにより、指定した位置を知ることができます。ただし、_trackcntl() では、スクロールバーは描き直さないので、アプリケーションが描き直さなければなりません。

ダイアログマネージャを使うと、COMMAND.func に以下の値が返ります。

PAGEU　なし (ボックスの上)
PAGED　なし (ボックスの下)
MOVEV　ドラッグ中、スクロールボックスが動きます。

- スクロールバーで使用する定数は以下のとおりです。

```
#define PAGEL   1       /* 左ページ */
#define PAGER   2       /* 右ページ */
#define PAGEU   3       /* 上ページ */
#define PAGED   4       /* 下ページ */
#define MOVEH   128     /* スクロールボックスを横に動かす */
#define MOVEV   129     /* スクロールボックスを縦に動かす */
```

- スクロールバーで使用する構造体は以下のとおりです。

```
typedef struct  _bartmp {
        WORD    curnum;         /* 現在の値 */
        WORD    maxnum;         /* 最大値 */
        WORD    pagenum;        /* ページの値 */
}       BARTMP;
```

カラーアイコン（ICON_CNTL）

書　式

```
static  TINY     pat8[ ] = { 0, 124, 70, 70, 70, 126
                 62, 0 };
static  CICON    icnmsg = { 1  255, pat8 };
static  CONTROL  cntl[ ] = { CIONC CNTL, sw, xp, yp,
                 xs, ys, (char *)icnmsg };
```

表　示　色を指定したパターンを表示します。

動　作　ドラッグ中は、リバースします。

カラーアイコンで使用される構造体は以下のとおりです。

```
typedef struct  _cicon {
        COLOR   on,
        COLOR   off
        TINY    *pat,
}       CICON;
```

アイコン（ICON_CNTL）

書　式

```
static  TINY     pat[ ] = {
                 1, 0, 2, 128, 4, 64, 9, 32,
                 16, 16, 36, 8, 72, 4, 144, 2,
                 64, 4, 32, 8, 16, 16, 8, 32
                 4, 64, 2, 128, 1, 0, 0, 0
                 };
static  CONTROL  cntl[ ] = { ICON_CNTL, sw, xp, yp, xs,
                 ys, (char *)pat };
```

表　示　xs×ysの大きさのパターンを表示します。

動　作　ドラッグ中は、リバースします。

フレーム (FRAME_CNTL)

書式

```
static  CONTROL   cntl[ ] = { FRAME_CNTL, sw, xp, yp
                  xs, ys, 0 };
```

表示 フレームを描きます。

動作 なし

ライン (LINE_CNTL)

書式

```
static  POS       theline[ ] = { { 0, 0 }, { 100, 100 }
                  };
static  CONTROL   cntl[ ] = { LINE_CNTL, sw, xp, yp, xs,
                  ys, (char *)theline };
```

表示 直線を描きます。

動作 なし

角の丸い四角 (ROUND_CNTL)

書式

```
static  CONTROL   cntl[ ] = { ROUND_CNTL, sw, xp, yp,
                  xs, ys, 0 };
```

表示 角の丸い四角を描きます。

動作 なし

塗りつぶし（ERASE_CNTL）

書　式

```
static  CONTROL  cntl[ ] = { ERASE_CNTL, sw, xp, yp,
                            xs, ys, (char *)WHITE };
```

表　示　領域を指定した色で塗りつぶします。

動　作　ドラッグ中は、リバースします。

文字列（STRING_CNTL）

書　式

```
static  CONTROL  cntl[ ] = { STRING_CNTL, sw, xp, yp,
                            xs, ys, "文字列" };
```

表　示　指定した位置に文字列を表示します。

動　作　ドラッグ中は、リバースします。

このコントロールは左下が原点

テキスト編集（TEXT_CNTL）

書　式

```
static  CONTROL  cntl[ ] = { TEXT_CNTL, sw, xp, yp, xs,
                            ys, (TEXT*)text };
```

表　示　指定した位置で文字列編集を行ないます。

動　作　クリックでテキストカーソルが移動し、ドラッグでテキスト内領域を指定します。テキストマネージャを簡単に利用するために使用します。

18.4 コントロールドライバの作成

コントロールドライバとは、ボタンやチェックマークなどのように、画面に部品を表示して、ユーザーからのアクションを得るための描画ルーチンです。コントロールドライバは、コントロールマネージャの下位ルーチンになります。

コントロールドライバは、コントロールマネージャの_setcntl() で、コントロールマネージャに登録され、ダイアログマネージャなどから、コントロールマネージャを経由して呼び出されます。コントロールドライバを作成するには、[A'] にバリエーション番号を入れ、他のレジスタに必要なパラメータを設定し、_drivecntl() をコールします。

表 3 24 コントロールドライバの種類

バリエーション番号	ファンクション名	名前	意味
0	drawcntl	DRAW_CD	コントロールの表示
1	selectcntl	SEL_CD	コントロールの選択
2	findpart	FIND_CD	パートの検索
3	catchlever	CATCH_CD	レバーの開始
4	drawlever	DLVR_CD	レバーの表示
5	eraselever	ELVR_CD	レバーの消去
6	freelever	FREE_CD	レバーの終了
7	openlever	OPEN_CD	コントロールのオープン
8	closelever	CLOSE_CD	コントロールのクローズ

コントロールドライバは以下の機能を満たさなければなりません。しかし、すべて満たす必要はなく、アプリケーションが必要な機能だけ作成し、その他はなにもせずにリターンするだけでもかまいません。

18.4.1 表記法

コントロールドライバの内容については、以下の表記で説明します。

コントロールドライバの名前

機能番号　コントロールドライバのバリエーション番号です。

書　式　コントロールマネージャから呼ばれる、コントロールドライバの書式です。

解　説　コントロールマネージャの解説です。

コントロールの表示 (DRAW_CD)

機能番号　0

書式

```
void     drawctrl(cp, part)
CONTROL  *cp;   [HL]
TINY     part;  [C]
```

解説

コントロールを画面上に描きます。part が 0 のときは、そのコントロール全部を、0 以外のときは、そのパートだけを描きます。

コントロールテンプレートの CONTROL.sw によって、表示形態が変わらなくてはいけませんが、その表現方法は各コントロールドライバの自由です。

ハイライト　　強調表示　　(例　リバース)
ディスエーブル　非アクティブ表示　(例　灰色で表示)

このルーチンは dispallcntl()、_dispcntl() で使用されます。

コントロールの選択 (SEL_CD)

機能番号　1

書式

```
void     selectcntl(cp, sw, part)
CONTROL  *cp;   [HL]
TINY     sw;    [E]
TINY     part;  [C]
```

解説

指定されたパートが選択されたこと（マウスでクリックされたときなど）を表示します。その表現方法は、各コントロールドライバの自由ですが、リバース表示が一般的です。

CONTROL.sw が 1 のときは選択を示す表示を、0 のときはそれの解除を行います。

part が 0 のときはそのコントロール全部を、0 以外のときはそのハートだけを強調します。CONTROL.sw は参照しなくてもかまいません。コントロールマネージャは偶数回呼びます。

このルーチンは _trackcntl()、_actioncntl() で使用されます。

パートの検索（FIND_CD）

機能番号 2

書式

```
TINY     findpart(cp, where)
CONTROL  *cp;      [HL]
POS      *where;   [OE]
```

解説

whereがどのパートに入っているかを調べます。パートが1つしかないときは、0以外を返すだけでかまいません。コントロールテンプレートで指定されているエリアのチェックは必要ありません。

レバーの検索（CATCH_CD）

機能番号 3

書式

```
void     catchlever(cp, where, part)
CONTROL  *cp;      [HL]
POS      *where;   [OE]
TINY     part;     [C]
```

解説

レバーをつかんだことをドライバに伝えます。

レバーの表示（DLVR_CD）

機能番号 4

書式

```
void     drawlever(cp, where, part)
CONTROL  *cp;      [HL]
POS      *where;   [OE]
TINY     part;     [C]
```

解説

レバーを描きます。例えば、スクロールボックスを動かすときのラバーバンド表示などで使われます。eraselever()と共通のルーチンでもかまいません。コントロールマネージャは偶数回呼びます。

レバーの消去（FLVR_CD）

機能番号　5

書式

```
void     eraselever(cp, where, part)
CONTROL  *cp;     [HL]
POS      *where;  [DE]
TINY     part;    [C]
```

解説　レバーを消します。例えば、スクロールボックスを動かすときのラバーバンド表示の消去などに使われます。drawlever() と共通のルーチンでもかまいません。コントロールマネージャは偶数回呼びます。

レバーの終了（FREE_CD）

機能番号　6

書式

```
BOOL     freelever(cp, where, part)
CONTROL  *cp;     [HL]
POS      *where;  [DE]
TINY     part;    [C]
```

解説　レバーを dpos の位置で離したことをドライバに伝えます。

コントロールのオープン（OPEN_CD）

機能番号　7

書式

```
STATUS   open(cp)
CONTROL  *cp;  [HL]
```

解説　コントロールドライバをオープンします。_opencntl() で使用されます。

コントロールのクローズ (CLOSE CD)

機能番号 8

書式

```
STATUS   close(cp)
CONTROL  *cp;  [HL]
```

解説 コントロールドライバをクローズします。_closecntl() で使用されます。

18.4.2 標準コントロールのカラー化

標準コントロールを任意の色で描画するときは、カスタムコントロールを作成します。

各標準ドライバは、カレントペンで描画を行うので、カスタムドライバは、標準ドライバを呼ぶ前にカレントペンを任意の色に設定することにより、標準コントロールのカラー化ができます。

カラーコントロールはメッセージを解析し、カレントペンを変更後、コントロールテンプレートを標準コントロール用に直して、標準コントロールドライバをコールします。コントロールマネージャがペンを保存するので、カラーコントロールはセーブは行わなくてもかまいません。

18.5 ファンクション一覧

コントロールマネージャには、以下のファンクションがあります。

表 3.25 コントロールマネージャのファンクション一覧

機能番号	[illegible]	意味	ページ
98	_initcntl()	コントロールマネージャの初期化	571
101	_setcntl()	カスタムコントロールの割り付け	571
102	opencntl()	コントロールのオープン	572
103	_findpart()	パートナンバーの獲得	572
104	_dispcntl()	コントロールの表示	573
105	_dispallcntl()	配列内のコントロールすべての表示	573
106	_trackcntl()	コントロールの実行	574
107	_actioncntl()	コントロール実行中の表示	574
108	_closecntl()	コントロールのクローズ	575
109	_openallcntl()	配列内のコントロールすべてのオープン	575
110	_testcntl()	任意の座標がコントロールに含まれるかの検索	576
111	_findcntl()	指定した座標を含むコントロールの検索	576
112	_drivecntl()	コントロールドライバの直接呼び出し	577

18.6 ファンクションの説明

以下では、コントロールマネージャの各ファンクションについて説明します。

18.6.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号 各ファンクションに割り当てられている番号です。

書式 各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- ファンクションの戻り値の型
- ファンクション名
- 引数
- 引数の意味
- 引数の受け渡しに使われるCPUレジスタ（アセンブラプログラミング時に使用）
- 引数名
- 引数の型

戻り値 そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- 戻り値の意味
- 戻り値
- 戻り値の引き渡しに使われるCPUレジスタ（アセンブラプログラム時に使用）

解説 そのファンクションがどのような動作をするかを示します。

コントロールマネージャの初期化

機能番号	98
書　式	void _initcntl(void)
戻り値	なし

解　説

コントロールマネージャを初期化します。カスタムドライバのクリア、標準コントロールなども初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

カスタムコントロールの割り付け

機能番号　101

書　式

```
HANDLE   setcntl(func, cntl)
TINY     (*func)()  [HL]  ドライバのアドレス
HANDLE   cntl       [E]   コントロールのハンドル
```

戻り値

[A]　成功した場合、割り付けられたコントロールのハンドル
　　失敗した場合、ERROR

解　説

新しいカスタムコントロールを割り付けます。カスタムコントロールドライバのアドレスfuncと、割り付けるハンドルcntlを渡します。ハンドルは64～127でなければなりません。cntlが0のときは、新しいハンドルを割り付けます。特に理由のない限り、新しいハンドルを割り当てて下さい。すでに割り当ててあるハンドルに割り付けることもできるので、オーバーレイ先で、間違って他のアプリケーションの割り付けを壊す可能性があります。

コントロールのオープン

機能番号	102

書式

```
STATUS    _opencntl(ctemp)
CONTROL   *ctemp  [HL]  CONTROL構造体へのポインタ
```

戻り値

[A] OK オープン成功
　　ERROR オープン失敗

解説

コントロールをコントロールテンプレートとメッセージにしたがってオープンし、そのコントロールを使用できるようにします。標準コントロールのほとんどはオープンしなくても使用できますが、初期化を必要とするようなコントロールドライバでは、このファンクションを使用します。

パート番号の獲得

機能番号	103

書式

```
TINY      findpart(ctemp, where)
CONTROL   *ctemp  [HL]  CONTROL構造体へのポインタ
POS       *where  [DE]  コントロールのローカル座標
```

戻り値

[A] 指定コントロールのパート番号
　　0が返った場合、どのパートにも属さない

解説

コントロールテンプレートおよび座標を渡すと、それに対応するコントロールのパート番号を返します。座標はそのコントロールのあるウィンドウのローカル座標です。このルーチンはあるコントロールが複数のパートで成り立っており、それをクリックする位置により処理が異なる場合に用います。

コントロールの表示

機能番号　104

書　式
```
void      dispcntl(ctemp, part)
CONTROL   *ctemp   [HL]  CONTROL構造体へのポインタ
TINY      part     [E]   パート番号
```

戻り値　なし

解　説　コントロールテンプレートとその中のメッセージにしたがって、ctempで指定されたコントロールを表示します。カレントペンとカレントフォントは表示前に保存され、コントロールドライバを呼び出して表示した後に回復されます。partが0のときは、そのコントロールをすべて描きます。

配列内のコントロールすべての表示

機能番号　105

書　式
```
void      _dispallcntl(ctemp)
CONTROL   *ctemp   [HL]  CONTROL構造体へのポインタ
```

戻り値　なし

解　説　ctempにコントロールテンプレートの配列の先頭アドレスを入れてこのルーチンを呼ぶと、CONTROL.swのFINが1になっているコントロールがあるまで、すべてのコントロールが表示されます。ただしCONTROL.swのMSKが1になっているコントロールは表示されません。

コントロールの実行

機能番号	106

書式

```
BOOL      _trackcntl(ctemp, part, event, (*func)())
CONTROL   *ctemp     [HL]  CONTROL 構造体へのポインタ
TINY      part       [A']  パート番号
EVENT     *event     [DE]  イベント構造体へのポインタ
TINY      (*func)()  [BC]  コールバックルーチンのアドレス
```

戻り値

[A] TRUE 正常に実行終了
FALSE 異常終了

解説

コントロールを実行します。1st ボタンが離されるまで制御は戻ってきませんが、このルーチンは1st ボタンが押されている間中、繰り返し行われる処理を行うルーチン（コールバックルーチン）の func を呼び続けます。コールバックルーチンのアドレスを渡すと、それに対応する動作をします。コールバックが必要でないときは、NULL (0000H) を渡します。パート番号が128以上のときは、レバーを示します。

コールバックルーチンへの引数は以下の通りです。

```
TINY      func(ctemp, pos, part)
CONTROL   *ctemp  [HL]  コントロールテンプレート
POS       *pos    [DE]  ローカル座標
TINY      part    [C]   パート番号
```

コントロール実行中の表示

機能番号	107

書式

```
void      _actioncntl(ctemp, sw, part)
CONTROL   *ctemp  [HL]  CONTROL構造体へのポインタ
TINY      sw      [E]
TINY      part    [C]
```

戻り値

なし

解説

コントロールが選択され、実行中であるということを示す表示を行います。sw が1のときは表示、0のときはは解除を行います。このルーチンはコントロールドライバの selectcntl() をコールします。

コントロールのクローズ

機能番号	108
書式	STATUS closecntl(ctemp) CONTROL *ctemp [HL] CONTROL 構造体へのポインタ
戻り値	[A] OK クローズ成功 ERROR クローズ失敗
解説	コントロールをクローズします。標準コントロールのほとんどはこのルーチンを実行しても何もしませんが、終了処理を必要とするようなコントロールドライバでは、このファンクションをコールしなければなりません。

配列内のコントロールすべてのオープン

機能番号	109
書式	STATUS _openallcntl(ctemp) CONTROL *ctemp [HL] CONTROL 構造体の配列へのポインタ
戻り値	[A] OK オープン成功 ERROR オープン失敗
解説	ctemp にコントロールテンプレートの配列の先頭アドレスを入れてこのルーチンをコールすると、CONTROL.sw の FIN が 1 になっているコントロールがあるまで、すべてのコントロールに対して、_opencntl() がコールされます。ただし、CONTROL.sw の MSK が 1 になっているコントロールに対してはコールされません。

任意の座標がコントロールに含まれるかの検索

機能番号　110

書　式

```
BOOL     _testcntl(ctemp, where)
CONTROL  *ctemp  [HL]  コントロール構造体へのポインタ
POS      *where  [DE]  ローカル座標
```

戻り値　[A]　TRUE　含まれる
　　　　　　FALSE　含まれない

解　説

　where で指定された任意の座標が、ctemp で指定されたコントロールに含まれているかどうかを調べます。座標はそのコントロールのあるウィンドウのローカル座標です。含まれていたら TRUE、そうでなかったら FALSE を返します。

指定した座標を含むコントロールの検索

機能番号　111

書　式

```
TINY     _findcntl(ctemp, where)
CONTROL  *ctemp  [HL]  コントロールの配列へのポインタ
POS      *where  [DE]  ローカル座標
```

戻り値　[A]　where が含んでいるコントロール配列内の要素番号
　　　　　どれにも含まれていなければ ERROR

解　説

　ctemp にコントロールテンプレートの配列の先頭アドレスを、where に座標を入れてこのルーチンを呼ぶと、どのコントロールにその座標が含まれているかを返します。座標はそのコントロールのあるウィンドウのローカル座標です。このルーチンは、_testcntl() をコントロールの数だけコールし、指定した位置がそのどれにも含まれていなければ、ERROR を返します。CONTROL.sw の DIS または MSK のどちらかが 1 であるコントロールは検索されません。

コントロールドライバの直接呼び出し

機能番号　112

書　式

```
TINY     _drivecntl(ctemp, where, part, vnum)
CONTROL  *ctemp  [HL]  CONTROL構造体へのポインタ
POS      *where  [DE]  コントロールのローカル座標
TINY     part    [BC]  パート番号
TINY     vnum    [A']
```

戻り値　[A]　ドライバの実行結果

解　説

　コントロールドライバを直接呼び出します。カレントペン、カレントフォントは保存しません。非常に低レベルなルーチンなので、扱いに注意して下さい。このルーチンの目的は標準コントロールをカラー化することですが、標準コントロールを使いやすくすることもできます。

19章 メニューマネージャ

この章では、メニューマネージャの構成や各ファンクションについて説明します。

19.1 メニューマネージャとは

メニューマネージャは、MSXViewの標準的なユーザーインターフェイスの1つである「文字列メニュー」をサポートするためのマネージャです。通常のアプリケーション開発では、ユーザーが扱い易いようなメニューの管理はたいへん面倒ですが、MSXViewでは、メニュー内に表示する文字列を所定のデータ形式で並べるだけで、メニューの大きさや配置にいたるまで、メニューマネージャが管理します。メニューはポインティングデバイスを使用してもキーボードを使用しても、簡単に選択できます。

メニューマネージャは、「メニューバー」、「ポップアップ」などのメニューを管理します。

基本的に、メニューは現在のカーソル位置を中心とした位置に表示されます。横は画面いっぱいに、縦はツールボックスなどに重ならないように、上端のY座標が16〜212までに自動的に調整されます。ポップアップの大きさはメニューテンプレートの内容により、自動的に最小の大きさに設定されます。表示するメニューの内容は、原則として文字列で構成されます。さらに、それぞれの項目に以下のような属性が指定できます。

- チェックマークをつける。
- 項目表示を禁止する。
- メニューを灰色にし選択できなくする。
- 両端に線をつける。
- 文字列をセンタリングする。
- 太字にする。
- 1行に複数の文字列を並べる。

また、キーボードを使ってワンタッチでメニューを選択できるようにするため、メニューの要素ごとにキーコードが指定できます。

メニューの役割は、ポインティングデバイスまたはキーボードによって、文字列項目の一覧の中から特定の項目を選び出すことです。メニュー自体はそれ以上の動作は行いません。

19.2 メニューマネージャの使い方

メニューマネージャを使うには、表示するメニュー形状を定義する「メニューテンプレート」を用意しておき、必要に応じて各種のルーチンをコールします。

19.3 メニューマネージャの構成と機能

メニューは標準的に、「タイトルバー」、「DA バー」、「コマンドバー」を開いておき、メニューがクリックされることによって、ポップアップがオープンします。

図 3.5 メニューバーの各部の名称

19.3.1 タイトルバー

タイトルバーは、文書ウィンドウに対する操作を選択するためのメニューで、通常画面の左上にオープンします。

ここには、ファイル処理、印刷、終了などのメニューが入っています。通常、タイトルバーには、現在編集中のファイル名を表示します。ただし、編集中のファイルが新規作成中の場合には、「新規」あるいはそのアプリケーションの名前を表示します。通常の起動時には、アプリケーション名を入れます。

一般的に、ここに入るメニューの内容は次のようなものです。

表 3.26 タイトルバーの内容

内容	意味
新規	編集内容を破棄して初期状態にします。
保存	編集内容に名前をつけて保存します。
更新	編集内容を読み込んだときの名前で保存します。
読込	保存してあったデータをディスクから読み込みます。
登録	標準形式のデータ交換ファイルを作ります。
組込	標準形式のデータ交換ファイルを組み込みます。
印刷	印刷を行います。
印刷形式	印刷形式の設定を行います。
終了	アプリケーションを終了し、VSHELL に戻ります。

19.3.2　DA（デスクアクセサリ）バー

DAバーは、デスクアクセサリを起動するためのメニューで、デスクアクセサリメニューが入っています。通常このメニューは、タイトルメニューのすぐ右にオープンされます。

DA バーの処理については、システムマネージャの_openda()、_closeda()、_drivesystem()を参照して下さい。アプリケーションは、これらのファンクションを呼び出すだけで、DA バーに関する処理は考慮する必要はありません。

19.3.3　コマンドバー

コマンドバーには、アプリケーションの各メニューが入っています。この部分のメニューの内容は、アプリケーションに依存します。アプリケーション独自の道具（機能）を選択するためのメニューで、画面の右上に表示します。

独自の道具（機能）に対するポップアップは　アプリケーションが定義し、必要な処理を行います。

19.3.4　ポップアップ

ポップアップは、文字列を中心にした項目を、マーカーによってポインティングデバイスキーボードどちらからでも簡単に選択できるようになっているメニューです。

ポップアップは、単独にオープンして選択することもできますが、通常は、メニューのテンプレートの中に定義し自動的に選択するようにします。現在のカーソル位置を中心とした位置に表示され、横は画面いっぱいに、縦はツールボックスなどに重ならないように、16〜211 ドットの大きさに自動的に制限されます。ポップアップの大きさは、テンプレートによって自動的に最小に設定されます。

ポップアップが表示された瞬間には、マーカーは表示されておらず、マウスカーソルが動いた瞬間に、カーソルに最も近いアイテムにマーカーが表示されます。ドラッグしなくてもマーカーはポインティングカーソルに追従します。ポインティングカーソルがメニューの外に出たときは、マーカーは消えます。マウスカーソルが止まっている状態で、カーソルキーが押されると、それにしたがってマーカーが移動します。その時点でポインティングデバイス（多くの場合、マウス）が動いたら、即座にポインティングカーソルの近くにマーカーが移動します。

メニューの選択は、マーカーが出ている状態で 1st ボタン（右ボタン）をクリックするか、[SELECT] キー、[↵] キー、[SPACE] キーを押します。キャンセルは、マーカーが消えている状態で 2nd ボタン（左ボタン）クリックするか、[STOP] キー または [ESC] キーを押します。

19.3.5 メニューテンプレート

メニューの表示には、以下の 2 種類のテンプレートを使用します。

ポップアップテンプレート

```
typedef etruct  _popup {
        TINY    head;           /* ヘッダ */
        char    keycode;        /* キーコード */
        char    *name;          /* 項目名 */
}       POPUP;
```

メニューテンプレート

```
typedef etruct  _menutp {
        TINY    head;           /* ヘッダ */
        char    keycode;        /* キーコード */
        char    *name;          /* 項目名 */
        POPUP   *temp;          /* ポップアップのテンプレート */
}       MENU;
```

表 3 27 メニューテンプレートの内容

[illegible]	サイズ	[illegible]
head	1 バイト	ヘッダ ビットごとに次に示す意味があります。 ビット　意味 7　チェックマーク　文字列の先頭にチェックマークを表示します。 　0　チェックマークなし 　1　チェックマークあり

名前	サイズ	意味	
		ビット	意味
		6	マスク　項目表示を禁止します。 0　項目表示 1　項目表示なし、選択不可能
		5	ディスエーブル　項目選択を禁止します。 マスクビットが立っているときは、マスクが優先します。 0　通常黒色で表示し、選択可能 1　通常灰色で表示し、選択不可
		4	線　項目の区切りを表現します。 0　通常 1　文字列をセンタリングし両側に線を描き、選択できなくします。
		3	センタリング　項目名をセンタリングします。 0　通常 1　項目名をセンタリングします。
		2	太字　項目名を太字で表示します。 0　通常 1　文字を太字にします。
		1	コンティニュー　この項目で改行をせずに、次の項目を横に続けて表示します。 0　改行します。 1　TAB を入れます。改行しません。
		0	エンド　テンプレートの最後の項目のこのビットを立てることで、メニューマネージャが項目の個数を知ることができます。 0　通常 1　最後の項目

名前	サイズ	意味
keycode	1バイト	キーコード 文字列の最後に表示し、このキーと GRAPH キーを同時に押す、あるいはファンクションキーを押すことで、この項目を選択できるように設定します。
name	2バイト	項目名 項目名文字列の先頭番地をここに入れておきます。表示のときに、この文字列が表示されます。 英文字などはプロポーショナル表示されます。日本語は2~3文字の項目が多いので、横方向に複数のアイテムを入れることができます。 文字列中に、TABコードや改行コードを入れることはできません。
temp	2バイト	テンプレート メニューバーでは項目ごとにポップアップを持っています。そのテンプレートアドレスを入れます。 0を入れておくとポップアップは表示せず、項目が選択されたことを直ちに返します。

テンプレートの例

```
/* テンプレートの例 */

static  MENU    tooltemp[ ] = {
                { NON, 0xf1, "切", 0 }
                { NON, 0xf2, "貼", 0 },
                { BLD | CHK, 0xf3, "文字の選択", &mojipop[0] },
                { FIN, 0xf4, "編集", \&editpop[0] }
}

/* ポップアップテンプレートの例 */

static  POPUP   mojipop[ ] = {
                { CNT | CHK | ULN, 'K', "下線" },
                { BLD | CHK, 'B', "太字" },
                { CNT | CHK, 'I', "斜体" }, { FIN | CHK, 'O', "輪郭" }
}

static  POPUP   editpop[ ] = {
                { CNT, 'E', "切取" }, { NON, 'P', "貼付" },
                { CNT, 'U', "取消" }, { NON, 'C', "複写" }
}
```

19.4 ファンクション一覧

メニューマネージャには、以下のファンクションがあります。

表 3.28 メニューマネージャのファンクション一覧

機能番号	名前	意味	ページ
150	_initmenu()	メニューマネージャの初期化	587
153	_openmenu()	メニューのオープン	587
154	_closemenu()	メニューのクローズ	588
155	_selectmenu()	メニュー項目が選択されたときの処理	588
156	_keymenu()	ショートカットキーの処理	589
157	_hilite()	メニュー項目の強調	589
158	_ismenu()	メニュー内のイベントのテスト	590
159	_selectpopup()	ポップアップの処理	590
160	_keypopup()	ポップアップでのショートカットキーの処理	591

19.5 ファンクションの説明

以下では、メニューマネージャの各ファンクションについて説明します。

19.5.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号　各ファンクションに割り当てられている番号です。

書　式　各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- STATUS — ファンクションの戻り値の型
- _clearwin — ファンクション名
- (win) — 引数
- ウィンドウハンドル — 引数の意味
- [A] — 引数の受け渡しに使われる CPU レジスタ (アセンブラプログラミング時に使用)
- win — 引数名
- HANDLE — 引数の型

戻り値　そのファンクションの動作の結果、どのような値が返されるかを示します。

解　説　そのファンクションがどのような動作をするかを示します。

メニューマネージャの初期化

機能番号　15[illegible]

書式
```
void  initmenu(void)
```

戻り値　なし

解説

メニューマネージャを初期化します。チェックマークパターンを設定したり、ハイライトやメニューハンドルをクリアします。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

メニューのオープン

機能番号　153

書式
```
HANDLE  _openmenu(area, style, win, pen, font, temp)
AREA    *area  [HL]   メニューを表示するエリア
TINY    style  [DE]   メニュースタイル
HANDLE  win    [C]    ウィンドウハンドル
HANDLE  pen    [A']   ペンハンドル
HANDLE  font   [E']   フォントハンドル
MENU    *temp  [BC']  MENU構造体
```

戻り値　[A]　メニューハンドル

解説

メニューとしてウィンドウを開き、テンプレートにしたがって項目名を表示し、メニューハンドルを返します。

win は特に理由のない限り 0 (NEW) にしておいて下さい。

style はウィンドウマネージャで使用されるウィンドウの形状と同じです。ビット 7 がセットされていれば FIX ウィンドウを使いますが、通常は FLOAT ウィンドウを使用します。

メニューのクローズ

機能番号　154

書式
```
STATUS  _closemenu(menu)
HANDLE  menu  [A]  メニューハンドル
```

戻り値　[A]　OK　　クローズ成功
　　　　　　ERROR　クローズ失敗

解説　ハンドルで指定されたウィンドウを削除し、メニューハンドルを解放します。ウィンドウマネージャの_deletewin()を使用すると、ウィンドウだけが削除され、メニューハンドルは解放されません。メニューが不必要になったら、必ずこのファンクションをコールして下さい。

メニュー項目が選択されたときの処理

機能番号　155

書式
```
BOOL     _selectmenu(com, where)
COMMAND  *com    [HL]  COMMAND構造体へのポインタ
POS      *where  [OE]  POS構造体へのポインタ
```

戻り値　[A]　TRUE　　選択された項目がある
　　　　　　FALSE　何も選択されなかった

解説　whereで指定した位置が、現在開かれているメニューに入っていれば、メニュー項目の強調とポップアップ表示をし、カーソルの動きを追って選択された項目番号をcomに格納します。ポップアップの外側がクリックされたときは、そのイベントをイベントキューに戻し（_ungetevent()する）FALSEを返します。TRUEが返ったときは、メニュー項目の強調がそのままになっているので、_hilite()をコールして、強調を解除して下さい。

ショートカットキーの処理

機能番号　156

書式

```
BOOL     _keymenu(com, where, keycode)
COMMAND  *com     [HL]  COMMAND構造体へのポインタ
POS      *where   [DE]  POS構造体へのポインタ
char     keycode  [C]   キーコード
```

戻り値　[A]　TRUE　項目が選択された
　　　FALSE　該当する項目がなく、選択されなかった

解説

与えられたキーコードを、メニューとポップアップのテンプレートから捜して選択します。項目が存在したときはその項目を強調します。

選択された項目がメニューのテンプレートにあったたときは、ポップアップを表示してカーソルをドライブし、その項目番号を返します。選択された項目がポップアップのテンプレートにあったときは、ただちにその項目番号を返します。

該当する項目がなかった場合には何もせず FALSE が返されます。TRUE が返ったときには、項目の強調はそのままになっているので、_hilite() をコールして強調を解除して下さい。

メニュー項目の強調

機能番号　157

書式

```
void     _hilite(com)
COMMAND  *com  [HL]  COMMAND構造体へのポインタ
```

戻り値　なし

解説

指定したメニューの項目を強調します。ポップアップの項目指定は無視されます。com が NULL のときは、強調を解除します。

メニュー内のイベントのテスト

機能番号　158

書式
```
BOOL    _ismenu(event)
EVENT   *event  [HL]  EVENT構造体へのポインタ
```

戻り値　[A]　TRUE　メニュー内でイベントがあった
　　　　　　FALSE　メニュー内ではなかった

解説　eventで指定したイベントが、メニュー内で起きたものかどうかを返します。

ポップアップの処理

機能番号　159

書式
```
TINY     selectpopup(temp, where, pen, font)
POPUP    *temp    [HL]
POS      *where   [DE]
HANDLE   pen      [A']
HANDLE   font     [E']
```

戻り値　[A]　選択された項目番号
　　　　　　何も選択されなかった場合0

解説　ポップアップをテンプレートにしたがって表示し、カーソルの動きを監視して、選択された項目番号を返します。また、ポップアップの外側がクリックされたときは、そのイベントをイベントキューに戻し、0で返ります。

ポップアップでのショートカットキー処理

機能番号	160

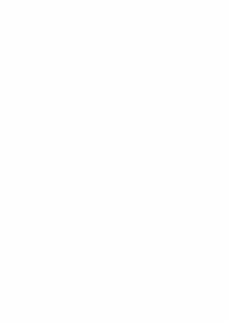
書式

```
TINY    _keypopup(temp, keycode)
POPUP   *temp     [HL]  POPUP構造体へのポインタ
char    keycode   [E]   キーコード
```

戻り値

[A] 選択された項目番号
　　何も選択されなかった場合0

解説

　ポップアップは表示せずに、テンプレートにしたがってキーコードをチェックし、選択された項目番号を返します。

20章 ダイアログマネージャ

この章では、ダイアログマネージャの構成や各ファンクションについて説明します。

20.1 ダイアログマネージャとは

ダイアログマネージャは、MSXView アプリケーションがユーザーからの応答を得るためのルーチン群です。

内部的には、ディスプレイマネージャとコントロールマネージャを呼び出して、ユーザーの選択を返します。

20.2 ファンクション一覧

ダイアログマネージャには、以下のファンクションがあります。

表 3.29 ダイアログマネージャのファンクション一覧

機能番号	名前	意味	ページ
257	_initdlg()	ダイアログマネージャの初期化	595
258	_opendlg()	ダイアログボックスの表示	595
259	_closedlg()	ダイアログボックスのクローズ	595
260	_dlgselect()	ダイアログ上のアクションの獲得	596
261	_modaldlg()	モーダルダイアログのアクションの獲得	596
262	_popupdlg()	ポップアップダイアログの表示とアクションの獲得	597
263	_message()	メッセージダイアログボックスの表示	597
373	_errmessage()	メッセージダイアログボックスの表示2	598

20.3 ファンクションの説明

以下では、ダイアログマネージャの各ファンクションについて説明します。

20.3.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号 各ファンクションに割り当てられている番号です。

書　式 各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```

- ファンクションの戻り値の型
- ファンクション名
- 引数
- 引数の意味
- 引数の受け渡しに使われる CPU レジスタ(アセンブラプログラミング時に使用)
- 引数名
- 引数の型

戻り値 そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```

- 戻り値の意味
- 戻り値
- 戻り値の引き渡しに使われる CPU レジスタ(アセンブラプログラム時に使用)

解　説 そのファンクションがどのような動作をするかを示します。

ダイアログマネージャの初期化

機能番号	257
書式	`void _initdlg(void)`
戻り値	なし
解説	ダイアログマネージャを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

ダイアログボックスの表示

機能番号　258

書式

```
HANDLE   _opendlg(area, style, win, pen, font, temp)
AREA     *area  [HL]   ダイアログボックスを表示するエリア
TINY     style  [E]    ダイアログボックスの形状
HANDLE   win    [C]    ウィンドウハンドル
HANDLE   pen    [A']   ペンハンドル
HANDLE   font   [E']   フォントハンドル
CONTROL  *temp  [BC']  コントロールへのポインタ
```

戻り値　[A]　ダイアログハンドル

解説　指定されたダイアログボックスを表示します。

ダイアログボックスのクローズ

機能番号　259

書式

```
void     _closedlg(dlog)
HANDLE   dlog  [A]  ダイアログハンドル
```

戻り値　なし

解説　dlogで指定したダイアログボックスをクローズします。

ダイアログ上のアクションの獲得

機能番号	260

書式

```
BOOL     _dlgselect(dlog, event, command)
HANDLE   dlog      [A]  ダイアログハンドル
EVENT    *event    [DE] EVENT構造体へのポインタ
COMMAND  *command  [BC] COMMAND構造体へのポインタ
```

戻り値

[A] TRUE 何か選択された
FALSE 何も選択されなかった

解説

_opendlg()で作成したダイアログへのユーザーの応答を返します。dlogには、_opendlg()で返ってきたハンドル番号をセットします。commandに_opendlg()でセットした、コントロールテンプレート中の選択されたアイテム（配列の要素番号）が返されます。

モーダルダイアログのアクションの獲得

機能番号	261

書式

```
BOOL     _modaldlg(dlog, event, command)
HANDLE   dlog      [A]  ダイアログハンドル
EVENT    *event    [DE] EVENT構造体へのポインタ
COMMAND  *command  [BC] COMMAND構造体へのポインタ
```

戻り値

[A] TRUE 何か選択された
FALSE 何も選択されなかった

解説

表示されたダイアログでのアクションを得て、その結果を返します。このファンクションは、_dlgselect()と同様の動作をしますが、内部ですべてのイベントを処理するので、アプリケーション側でイベントを取得して渡す必要はありません。

ポップアップダイアログの表示とアクションの獲得

機能番号　262

書式
```
TINY     popupdlg(area, style, center, pen, font, temp)
AREA     *area   [HL]   ダイアログボックスを表示するエリア
TINY     style   [E]    ダイアログボックスの形状
POS      center  [BC]   表示位置
HANDLE   pen     [A']   ペンハンドル
HANDLE   font    [E']   フォントハンドル
CONTROL  *temp   [BC']  コントロールへのポインタ
```

戻り値　[A]　ERROR 以外　選択されたアイテム
　　　　　　ERROR　　　何も選択されなかった

解説　centerで指定された位置を中心として、画面におさまるように自動的に表示位置を調整したダイアログを表示し、ユーザーのアクションを得てその結果を返します。結果はコントロールテンプレートのアイテム（配列の要素番号）で返されます。

メッセージダイアログの表示

機能番号　263

書式
```
TINY  message(msg, licon, ricon)
char  *msg    [HL]  表示する文字列へのポインタ
char  *icon1  [DE]  アイコンパターンへのポインタ
char  *icon2  [BC]  アイコンパターンへのポインタ
```

戻り値　[A]　0　文字列部分がクリックされたとき
　　　　　　1　icon1 がクリックされたとき
　　　　　　2　icon2 がクリックされたとき

解説　メッセージダイアログを表示します。msg が表示される文字列、licon が左側に表示されるアイコン、riconが右側に表示されるアイコンです。

メッセージダイアログの表示2

機能番号 373

書式

```
TINY _errmessage(msg, icon, htn)
char *msg     表示する文字列へのポインタ
char *icon    アイコンパターンへのポインタ
char *btn[]   ボタン文字列の配列へのポインタ
```

戻り値 [A] ボタン番号

解説

アイコンと文字列およびボタンを表示して、どのボタンが左クリックされたかを、ボタンの番号で返します。与えるボタン文字列の配列の最後は、終了を意味するためにNULLを置きます。ボタンは1~3です。

右クリックされると、最大のボタン番号を返します。

例えば2つのボタン文字列の配列は以下のようにします。

```
char *htn[] = { "ボタン1", "ボタン2", NULL };
```

この場合、ボタン1が左クリックされると0が、ボタン2が左クリックされると1が返ります。マウスカーソルがどこであれ 右クリックされると1を返すので、一番最後のボタンは「中止」などを選択するために使用します。

21章 その他のマネージャ

この章では、今まで解説した以外のマネージャの各ファンクションについて説明します。その他のマネージャとしては、以下のものがあります。

- システムマネージャ
- キーマップマネージャ
- サウンドマネージャ
- メモリマネージャ
- プリントマネージャ

21.1 ファンクション一覧

その他のマネージャには、以下のファンクションがあります。

表 3.30 システムマネージャのファンクション一覧

機能番号	[illegible]	[illegible]	ページ
1	_wr_sysdata()	システムデータの書き込み	602
6	_rd_sysdata()	システムデータの読み出し	602
293	_openda()	DA メニューのオープン	603
294	_closeda()	DA メニューのクローズ	603
295	_drivesystem()	DAの処理	603
301	_endview()	MSXView の終了	604
307	_setdate()	日付の設定	604
308	_getdate()	日付の獲得	604
309	_settime()	時刻の設定	605
310	_gettime()	時刻の獲得	605
372	_showtitle()	タイトルの表示	605
410	_dosexec()	DOS コマンドの実行	606

表 3.31 キーマップマネージャのファンクション一覧

機能番号	名前	意味	ページ
285	_initkeymap()	キーマップマネージャの初期化	607
286	_getkeyfunc()	機能コードの獲得	607
287	_mapcntlkey()	機能コードのマッピング	608
288	_getalphkey()	機能コードがマッピングされているキーの獲得	610
289	_getjekyfunc()	機能コードの獲得	610
291	_setkeymap()	キーマップの設定	610
292	_getkeymap()	キーマップの獲得	611

表 3.32 サウンドマネージャのファンクション一覧

機能番号	名前	意味	ページ
368	_pcmplay()	PCM の再生	612
369	_pcmrec()	PCMの録音	612

表 3.33 メモリマネージャのファンクション一覧

機能番号	名前	意味	ページ
303	_initmemory()	メモリマネージャの初期化	614
304	_sbrk()	データ領域の大きさの変更	614
305	_free()	メモリブロックの解放	615
306	_malloc()	メモリブロックの獲得	615
389	_openvb()	VRAM バッファのオープン	615
390	_closevb()	VRAM バッファのクローズ	616
391	_writevb()	VRAM バッファへの書き込み	616
392	_readvb()	VRAM バッファからの読み出し	617

表 3.34 プリントマネージャのファンクション一覧

機能番号	名前	意味	ページ
311	_initprint()	プリントマネージャの初期化	624
312	_chpd()	プリンタドライバの変更	624
313	printinfo()	プリント情報の獲得	624
314	_pd()	プリンタドライバの起動	625

21.2 ファンクションの説明

以下では、その他のマネージャの各ファンクションについて説明します。

21.2.1 表記法

ファンクションの説明では、次のように表記します。

ファンクションの機能を示します

機能番号　各ファンクションに割り当てられている番号です。

書式　各ファンクションを使用するときの書式を示します。

```
STATUS  _clearwin(win)
HANDLE  win  [A]  ウィンドウハンドル
```


戻り値　そのファンクションの動作の結果、どのような値が返されるかを示します。

```
[A]  OK     オープン成功
     ERROR  オープン失敗
```


解説　そのファンクションがどのような動作をするかを示します。

21.2.2 システムマネージャ

システムマネージャは、どのマネージャにもあてはまらないルーチン群です。

システムデータの書き込み

機能番号　1

書式

```
void      wr_sysdata(address, data)
unsigned  address  [HL]  システムデータファイルのアドレス
char      data     [E]   書き込むデータ
```

戻り値　なし

解説　MSXViewシステムデータエリアのaddressで指定したアドレスに、dataで指定したデータを書き込みます。システムデータエリアにはパレット情報やポインティングデバイスの種類など、MSXViewが内部的に参照する情報がおさめられています。システムデータエリアの内容は、MSXView起動時に「pref.mv」ファイルから読み込まれ、終了時に書き出されます。システムデータエリアの内容およびアドレスは将来変更される可能性があるので、各情報を参照・変更するときには、個別のアクセス用ファンクションを使用して下さい。

システムデータの読み出し

機能番号　6

書式

```
char      rd_sysdata(address)
unsigned  address  [HL]  アドレス
```

戻り値　[A]　読み出したデータ

解説　MSXViewシステムデータエリアの、addressで指定したアドレスからデータを読み出します。

DA メニューのオープン

機能番号	293

書式

```
HANDLE   openda(x, y)
int      x  [HL]  横方向のグローバル[illegible]標
int      y  [DE]  縦方向のグローバル[illegible]標
```

戻り値	[A]　DA メニューのハンドル

解説	x、y の位置を左上として、DA メニューをオープンします。

DA メニューのクローズ

機能番号	294

書式

```
void  _closeda(void)
```

戻り値	なし

解説	DA メニューをクローズします。

DA の処理

機能番号	295

書式

```
BOOL    _drivesystem(event, func)
EVENT   *event      [HL]  EVENT構造体へのポインタ
void    (*func)()   [DE]  コールバックルーチンへのポインタ
```

戻り値	[A]　常に TRUE

解説

DA メニューがクリックされたときの処理一切を行います。具体的には、カレントディレクトリおよび VIEWDA環境変数で指定したディレクトリに存在するデスクアクセサリプログラムをポップアップメニューに表示しユーザーの選択したデスクアクセサリを起動します。DA を起動する前に func をコールするので、DA に制御が移る前にアプリケーションが何か処理を行いたいときには、func に関数へのポインタを設定して下さい。特に処理する必要がなければ、func には NULL ポインタを設定して下さい。

MSXView の終了

機能番号	301
書式	`void _endview(void)`
戻り値	呼び出し元には戻りません
解説	MSXView を終了して、MSX-DOS2 に戻ります。終了前に MSXView カーネル内部のデータは、以下の環境変数で示すディレクトリにある「PREF.MV」に保存されます。

MSXView のバージョン	環境変数
1.0	VIEW
1.1	VIEW
1.2	VIEWDATA

日付の設定

機能番号	307
書式	`void _setdate(date)` `DATE *date [HL] DATE構造体へのポインタ`
戻り値	なし
解説	date で指定した日付を、現在の日付として設定します。

日付の獲得

機能番号	308
書式	`void _getdate(date)` `DATE *date [HL] DATE構造体へのポインタ`
戻り値	なし
解説	date に現在の日付を返します。

時刻の設定

機能番号　309

書　式

```
void  _settime(time)
TIME  *time  [HL]  TIME構造体へのポインタ
```

戻り値　なし

解　説　timeで指定した時刻を、現在の時刻として設定します。

時刻の獲得

機能番号　310

書　式

```
void  _gettime(time)
TIME  *time  [HL]  TIME構造体へのポインタ
```

戻り値　なし

解　説　timeに現在の時刻を返します。

タイトルの表示

機能番号　372

書　式

```
void  _showtitle(void)
```

戻り値　なし

解　説　MSXViewの起動時に表示されるタイトルを表示します。

DOSコマンドの実行

機能番号	410
書式	void _dosexec(cmd) char *cmd [HL] COMMAND2.COM に渡す実行コマンド行
戻り値	呼び出し元には戻りません

解説

MSXView から COMMAND2.COM を呼び出して、cmd で指定された DOS コマンドを実行します。cmdが指定するコマンド行は、127バイト以下でなければなりません。このファンクションは呼び出し元には戻らず、VSHELL に戻るので、これを呼び出すアプリケーションプログラムは、呼び出し前に終了処理を行わなくてはなりません。

COMMAND2.COMが呼び出される前には、MSXView を終了させるときと同じ処理が行われます。スクリーンモードは、MSXView が起動したときのモードに戻します。

DOS コマンドの実行が終ると、

```
Push any key to return to MSXView.
```

と表示し、何かキーを押すと、MSXView の VSHELL に戻ります。

MSXViewがCOMMAND2.COM を呼び出すときは、環境変数のSHELL を参照しているので、SHELL を別に設定すると、そのプログラムを呼び出すことができます。

また、SHELL が設定されていないときは、「A:¥COMMAND2.COM」が呼び出されます。MSXView が DOS コマンド cmd を渡すときは、81H〜0FFH 番地にその内容を、80H番地に文字数を格納しています。

このファンクションは、MSXViewversion 1.20以上で使用できます。

21.2.3 キーマップマネージャ

キーマップマネージャは、キーマップを管理するマネージャです。

キーマップマネージャの初期化

機能番号 285

書式

```
void    initkeymap(void)
```

戻り値 なし

解説

キーマップマネージャを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

機能コードの獲得

機能番号 286

書式

```
TINY    _getkeyfunc(event)
EVENT   *event   [HL]  イベント構造体へのポインタ
```

戻り値

[A] 機能コードを返します。ただし、以下の場合は、0を返します

- 与えられたイベントがキーボードイベントではないとき
- 特殊キーではなく、[CTRL] キーも押されていないとき
- [CTRL] キーとの組み合せに、何もマップされていないとき

解説

与えられたキーイベントに対して、マップされている機能コードを返します。ただし、特殊キー（[INS]、[DEL]、[BS]、[↵] など）には、あらかじめ特定の機能コードがマップされているので、特殊キーイベントを与えるとその機能コードが返ります。これにより、[↵] キーの代わりに、[CTRL] + [M] キーを使用したり、[BS] キーの代わりに、[CTRL] + [H] キーを使用することもできます。~~また、機能コードは、[CTRL] キーと任意のキーの組み合せに対して、mapcntkey() で設定することができます。~~

機能コードのマッピング

機能番号	287
書式	STATUS _mapcntlkey(keymap, funccode) TINY keymap [HL] TINY funccode [DE]
戻り値	[A] OK 設定成功 ERROR 設定失敗
解説	特定のキーと CTRL キーとの組み合せに対して、テキスト編集のための機能をマッピングします。一般的なシステムでは、CTRL キーと英字キーとの組み合せで、いろいろな編集機能が使えるようになっています。しかし、MSXView では、CTRL + 英字キー、CTRL + 数字キー、CTRL [illegible]ションキー、CTRL + 特殊キーの組み合わせ[illegible]います。 編集機能には、以下のようなファンクションコードが割り当てられています。以下は、標準インクルードファイルの<keydefs.h>内で定義されている内容です。

編集ファンクションの定義

```
/*
**                         カーソル移動関係
*/
#define CURSORUP           0x01    /* カーソル上移動 */
#define CURSORDOWN         0x02    /* カーソル下移動 */
#define CURSORLEFT         0x03    /* カーソル左移動 */
#define CURSORRIGHT        0x04    /* カーソル右移動 */
#define CURSORTOL          0x05    /* カーソル行頭移動 */
#define CURSOREOL          0x06    /* カーソル行末移動 */
#define CURSORHOME         0x07    /* カーソルホーム */
#define CURSORPAGEUP       0x08    /* カーソル1ページ上移動 */
#define CURSORPAGEDOWN     0x09    /* カーソル1ページ下移動 */
#define CURSORTOPOFTEXT    0x0a    /* カーソルをテキスト先頭へ移動 */
#define CURSORENDOFTEXT    0x0b    /* カーソルをテキスト終端へ移動 */
#define CURSORLINEUP       0x0c    /* 画面を1行上移動 */
#define CURSORLINEDOWN     0x0d    /* 画面を1行下移動 */
#define REWRITESCREEN      0x0e    /* 画面再表示 */
/*
**                         挿入と削除
*/
```

```
#define TOGGLEINSERT    0x10    /* 挿入モードの切り換え */
#define DELETELEFT      0x11    /* カーソル左削除 */
#define DELETERIGHT     0x12    /* カーソル右削除 */
#define INSERTLINE      0x13    /* 1行挿入 */
#define DELETELINE      0x14    /* 1行削除 */
#define DELETETOEOL     0x15    /* カーソルから行末までの削除 */
#define DELRANGE        0x16    /* レンジ選択されている場所を削除する */
/*
**                      その他の機能
*/
#define TOGGLESELECT    0x80    /* レンジ選択状態に入る・終了する (トグル)  */
#define STARTSELECT     0x81    /* レンジ選択状態に入る */
#define STOPSELECT      0x82    /* レンジ選択状態を終了する */
#define SELECTCHAR      0x83    /* カーソル位置の1文字をレンジ選択する */
#define SELECTWORD      0x84    /* カーソル位置の1単語をレンジ選択する */
#define SELECTLINE      0x85    /* カーソル位置の1行をレンジ選択する */
#define SELECTPARAGRAPH 0x86    /* カーソル位置の1章をレンジ選択する */
#define CUTTEXT         0x87    /* レンジ選択されているテキストを切りとる */
#define COPYTEXT        0x88    /* レンジ選択されているテキストを複写する */
#define PASTETEXT       0x89    /* カーソル位置にペーストバッファを貼る */
#define SETCURSOR       0x8a    /* カーソルの表示座標をcursorPtrに合わせる */
#define SETSCREEN       0x8b    /* dispTopLineに合わせて画面を書き直し、*/
                                /* カーソルが画面外に出たときには修正する */
/*
**                      特殊キー
*/
#define RETURNKEY       0x90    /* リターンキー */
#define SFTRETKEY       0x91    /* シフトキー + リターンキー */
#define SPACEKEY        0x92    /* スペースキー */
#define SFTSPCKEY       0x93    /* シフト + スペースキー */
#define ADVTEXT         0x94    /* 対象テキストを進める (TABキーと同じ) */
#define ESCKEY          0x95    /* ESC キー */
#define BACKTEXT        0x96    /* 対象テキストを戻す */
#define KANASPACE       0x97    /* カナシフト状態のスペースキー */
#define SELECTKEY       0x98    /* SELECT キー */
#define TABKEY          0x99    /* TAB キー */
```

ファンクションコードとして0を与えると、そのキーと CTRL キーとを一緒に押しても、何も機能しなくなります。また、ファンクションコードとして128～240を与えると、カスタムファンクションとなり、アプリケーションで自由にいろいろな機能をコントロールキーにマップすることができます。ただし、メニュー内の項目に対しては、メニューテンプレート内のショートカットの情報の最上位ビットを立てることによって、GRAPH キーのみならず、CTRL キーの使用が可能となっています。後者を使用することを推奨します。

機能コードがマッピングされているキーの獲得

機能番号 288

書式

```
TINY  getalphkey(funccode)
TINY  funccode  [A]  機能コード
```

戻り値 [A] ビット0〜6に英数字記号のASCIIコード
ビット7に CTRL キーの状態 (0=押下なし、1=押下あり)

解説 funccodeで指定した編集機能が、どのキーに割り付けられているかを返します。

機能コードの獲得

機能番号 289

書式

_getjkeyfunc(event)

```
TINY   _getjekyfunc(event)
EVENT  *event  [HL]  イベント構造体へのポインタ
```

戻り値 [A] 機能コード

解説 かな漢字変換中のキーイベントに対応する機能コードを返します。

キーマップの設定

機能番号 291

書式

```
void  _setkeymap(table)
char  *table  [HL]  キーマップテーブルへのポインタ
```

戻り値 なし

解説 キーマップをtableで指定したように設定します。

キーマップの獲得

機能番号　292

書　式

```
void   _getkeymap(table)
char   *table   [HL]   キーマップテーブルが返される領域
```

戻り値　なし

解　説

table で指定したメインメモリ上の領域に、キーマップテーブルを返します。

21.2.4　サウンドマネージャ

サウンドマネージャは、PCM の録音・再生を管理するマネージャです。

PCM の再生

機能番号　368

書式

```
STATUS  _pcmplay(pcm)
PCM     *pcm  [HL]  PCM構造体へのポインタ
```

戻り値

```
[A]  OK      最後まで正常に再生できたとき
     ERROR   [illegible]に STOP キーが押され[illegible]き
```

解説

PCM の再生をします。pcm には、PCM 構造体へのポインタを指定します。再生途中に STOP キーが押されたときは、ERROR が返ります。正しく最後まで再生できたときは、OK が返ります。どちらの場合でも、最後に再生された次のアドレスが PCM 構造体の start に、残りの長さが length に返されます。フラグは壊されます。

PCM の録音

機能番号　369

書式

```
STATUS  pcmrec(pcm)
PCM     *pcm  [HL]  PCM構造体へのポインタ
```

戻り値

```
[A]  OK      最後まで正常に録音できたとき
     ERROR   録音途中に STOP キーが押されたとき
```

解説

PCM の録音をします。pcm には PCM 構造体へのポインタを指定します。録音する領域、録音周波数などの PCM 構造体は、_pcmplay と同じです。録音途中に STOP キーが押されたときは、ERROR が返ります。正しく最後まで録音できたときは、OK が返ります。どちらの場合でも最後に録音された次のアドレスが PCM 構造体の start に、残りの長さが length に返されます。フラグは壊されます。

PCM 構造体は、次のようになっています。

```
typedef struct  _pcm {
        TINY    flag;           /* フラグ */
        long    start;          /* 開始アドレス */
        long    length;         /* 長さ（バイト） */
}       PCM;
```

フラグの設定値は、次のようになっています。

```
#define PCM_15K          0      /* 15.75  KHz */
#define PCM_8K           1      /*  7.875 KHz */
#define PCM_5K           2      /*  5.25  KHz */
#define PCM_4K           3      /*  3.9375KHz */

#define PCM_RAM          0      /* main RAM */
#define PCM_VRAM         128    /* VRAM */

#define PCM_COMP         4      /* compression */
#define PCM_NOCOMP       4      /* no compression */

#define SHIFT_TRIG_PCM   3      /* number of shift bit */
                                /* 0から15までのトリガレベルを flagに */
                                /* or するときのシフト値 */
```

以上の構造体およびフラグ設定値はPCM.Hで定義されています。

21.2.5 メモリマネージャ

メモリマネージャは、メモリを管理するマネージャです。

メモリマネージャの初期化

機能番号	303
書式	void initmemory(endap) char *_endap [HL] アプリケーションの最終アドレス
戻り値	なし
解説	_malloc()、free()、_sbrk()のメモリマネージャファンクションを使用するときは、このファンクションをコールしてメモリマネージャを初期化して下さい。

データ領域の大きさの変更

機能番号	304
書式	char * sbrk(block) int block [HL] 必要とするメモリブロックのサイズ
戻り値	[HL] メモリブロックへのポインタ 十分なメモリが獲得できなかった場合には-1
解説	blockで指定したバイト数のメモリブロックを割り当て、そのブロックへのポインタを返します。アプリケーション中で、このファンクションで獲得したメモリブロックは、そのアプリケーションが終了するまで存在し続けます。一時的に作業用メモリが必要なときは、_malloc()を使用すると必要がなくなった時点でメモリブロックを解放することができます。 アプリケーションでは、_malloc()を使って下さい。

メモリブロックの解放

機能番号　305

書式
```
void  free(block)
char  *block  [HL]  メモリブロックへのポインタ
```

戻り値　なし

解説　malloc() で獲得したメモリブロックを解放します。

メモリブロックの獲得

機能番号　306

書式
```
char  *_malloc(block)
int   block  [HL]  ブロックサイズ
```

戻り値　[HL]　メモリブロックへのポインタ
指定された量のメモリが獲得できなかった場合には-1

解説　block で指定したバイト数のメモリブロックを割り当て、そのブロックへのポインタを返します。

VRAMバッファのオープン

機能番号　389

書式
```
unsigned  _openvb(void)
```

戻り値　[HL]　使用可能な VRAM バッファのバイト数

解説　VRAM バッファをオープンします。スクリーンモードが 5 または 6 のときに、VRAM のページ3、4 を 64K バイトのバッファとして確保し、アプリケーションが自由に使用できるようにします。アプリケーションが VRAM バッファを使用していることがあるので、デスクアクセサリプログラムから VRAM バッファを使ってはいけません。また、VRAM バッファを使用中にスクリーンモードを変更すると、VRAM バッファの内容が壊れてしまいます。

VRAMバッファのクローズ

機能番号 390

書式

```
void _closevb(void)
```

戻り値 なし

解説 VRAMバッファをクローズします。

VRAMバッファへの書き込み

機能番号 391

書式

```
unsigned _writevb(src, dst, n)
char      *src  [HL]  メインメモリ上のアドレス
char      *dst  [DE]  VRAMバッファの先頭からのオフセット
unsigned  n     [BC]  書き込むバイト数
```

戻り値 [HL] VRAMバッファに書き込まれたバイト数
書き込みに失敗した場合0

解説 メインRAM上のデータをVRAMへ転送します。srcに転送元のメインメモリ上の先頭アドレスを、dstにVRAMバッファの先頭からのオフセットを、nに転送するサイズを指定します。

VRAMバッファからの読み出し

機能番号　392

書式

```
unsigned _readvb(src, dst, n)
char     *src  [HL]  VRAMバッファの先頭からのオフセット
char     *dst  [DE]  読み込み先メインメモリのアドレス
unsigned n     [BC]  読み込むバイト数
```

戻り値　[HL]　メインメモリ上に読み込まれたバイト数
読み込みに失敗した場合0

解説　VRAM上のデータをメインRAMへ転送します。srcに転送元のVRAMバッファの先頭からのオフセット、dstに読み込み先のメインメモリ上のアドレス、nに転送するサイズを指定します。

21.2.6 プリントマネージャ

プリントマネージャは、プリンタ出力を管理するマネージャです。プリンタマネージャは、指定されたデータをプリンタドライバに渡し、プリンタドライバが、本体に接続されているプリンタに応じたプリンタ制御コマンドをプリンタに送ります。

21.2.7 プリンタドライバ

プリンタドライバとは、プリンタとアプリケーションの間に入り、どのようなプリンタが接続されているかを、アプリケーションが意識しなくても印刷できるようにするためのオーバーレイプログラムです。

プリンタドライバはプリンタごとに用意され、ユーザーが「プリンタ.DA」を使用して接続されたプリンタ用のドライバを設定します。

アプリケーションが pd() を実行すると、現在設定されているプリンタドライバが呼び出されるので、どのプリンタが接続されているか(どのプリンタドライバが設定されているか)をアプリケーションは意識しないで印刷することができます。

21.2.8 プリンタドライバの変更

chpd() により、アプリケーションプログラムから、どのプリンタドライバを使用するかを設定できます。しかし、通常は「プリンタ.DA」によってユーザーが設定するので、むやみに変更してはいけません。

21.2.9 プリンタドライバの呼び出し

プリンタドライバを呼び出すには _pd() を使います。_pd() には機能番号を渡して、どのような処理をするかを指定します。機能番号とその内容は以下のとおりです。

表 3.35 プリンタドライバの機能番号

機能番号	名前	処理内容
0	PD_INIT	プリンタドライバの初期化
1	PD_OPEN	プリンタの印刷開始宣言
2	PD_CLOSE	プリンタの印刷終了宣言
3	PD_PRINT	印刷の実行
4	PD_MENU	用紙の設定
5	PD_START	開始ページ 終了ページ 印刷枚数の設定
6	PD_PAGE	印刷開始メッセージの表示

アプリケーションは以下の手順で印刷します。

1. PD_INIT でプリンタドライバを初期化します。
2. PD_MENU で用紙を設定します。

   PD_MENU では プリンタドライバがダイアログを表示し、ユーザーの設定を記憶します。用紙の設定は、アプリケーションのメニューで印刷の前に行なってもかまいません。
3. PD_START で開始ページ、終了ページ、印刷枚数を設定します。

   PD_MENU では、プリンタドライバがダイアログを表示し、ユーザーの設定を記憶します。
4. PD_PAGE で用紙セットのメッセージを表示します。
5. PD_OPEN で印刷の開始を宣言します。
6. PD_PRINT で印刷を実行します。

   印刷開始ページや終了ページ、印刷枚数などはアプリケーションが_printinfo() を呼び出して PRINT 構造体へのポインタを取得し、構造体内部の startp、endp、copy を調べて決定します。構造体内部の opt の最下位ビットが 1 のときは、カット紙の設定になっているので、アプリケーションは 1 ページ印刷するごとに、PD_PAGE を呼び出して用紙のセットの表示をしなくてはなりません。
7. PD_CLOSE で印刷の終了を宣言します。

プリンタドライバの機能

プリンタドライバの各機能について、以下で説明します。

表 3.36 プリンタドライバの機能

名前	機能
PD_INIT	プリンタドライバを初期化します。MSXView カーネル内部にあるプリンタ用のワークエリアを、プリンタドライバが持っている値で初期化します。_pd() は正しく初期化できた場合 OK を、初期化できなかった場合 OK 以外を返します。
PD_OPEN	印刷の開始を宣言します。プリンタをリセットし、印刷ができる状態にします。_pd() は印刷が可能になったら OK を返します。プリンタが接続されていなかったり、プリンタがオンラインでなかったり、用紙がなかったりしたら、OK 以外を返します。
PD_CLOSE	印刷の終了を宣言します。_pd() は正しく終了できた場合に OK を、終了できなかった場合 OK 以外を返します。

<table>
<tr><th>名前</th><th>機能</th></tr>
<tr><td>PD_PRINT</td><td>印刷を行ないます。PRINT構造体の buff で示されるバッファの内容を解釈し、プリンタの制御コードに変換してプリンタに送ります。buffは、通常は MAIL を示しています。MAIL は 080H～0FFFH 番地までの 128 バイトしかないので、アプリケーションがもっと多くのデータを一度に印刷したいときは変更することができます。変更するときは、buff の示す番地がプリンタドライバと重ならないように十分大きくなければなりません。変更したときは、印刷が終了したら buff が MAIL を示すように戻して下さい。
データの終わりは 00H をおきます。
印刷中に [GRAPH] + [STOP] が押されると、印刷を中断してアプリケーションに戻ります。
pd() は、印刷が正常に終了したら OK を、途中で中断されたら OK 以外を返します。</td></tr>
<tr><td>PD_MENU</td><td>用紙の設定をユーザーが行ないます。プリンタドライバがダイアログを表示し、ユーザーの設定を_printinfo() が返す構造体へのポインタで示される領域に記憶します。
設定される値（PRINT 構造体内部）は以下のとおりです。
<table>
<tr><th>構造体のメンバ</th><th>意味</th></tr>
<tr><td>opt（ビット 0）</td><td>用紙の種類
0 連続紙
1 カット紙</td></tr>
<tr><td>pid</td><td>用紙の ID
0 8×11 インチ
1 A4
2 B5
3 A5
4 B6
5 葉書</td></tr>
<tr><td>width</td><td>用紙の印刷可能な幅（ドット数）</td></tr>
<tr><td>hight</td><td>用紙の印刷可能な高さ（ドット数）</td></tr>
</table>
用紙の設定は、アプリケーションのメニューなどで印刷の前に行なってもかまいません。
_pd() は、設定されたら 0 を、中止されたら 1 を返します。</td></tr>
</table>

<table>
<tr><th>名前</th><th>機能</th></tr>
<tr><td>PD_START</td><td>開始ページ、終了ページ、印刷枚数を設定します。プリンタドライバがダイアログを表示し、ユーザーの設定を _printinfo() が返す構造体へのポインタで示される領域に記憶します。
設定される値（PRINT構造体内部）は以下のとおりです。
<table>
<tr><th>構造体のメンバ</th><th>意味</th><th></th></tr>
<tr><td>startp</td><td>開始ページ</td><td></td></tr>
<tr><td></td><td>0</td><td>最初のページから</td></tr>
<tr><td></td><td>それ以外</td><td>そのページから</td></tr>
<tr><td>endp</td><td>終了ページ</td><td></td></tr>
<tr><td></td><td>0</td><td>最後のページまで</td></tr>
<tr><td></td><td>それ以外</td><td>そのページまで</td></tr>
<tr><td>copy</td><td>印刷枚数</td><td></td></tr>
<tr><td></td><td>0</td><td>1枚のみ</td></tr>
<tr><td></td><td>それ以外</td><td>その枚数</td></tr>
</table>
pd() は、設定されたら 0 を、中止されたら 1 を返します。</td></tr>
<tr><td>PD_PAGE</td><td>用紙セットのメッセージを表示します。プリンタドライバは
「用紙をセットして RET キーを押して下さい」
[illegible]キーが押[illegible]
が押されたら 0 を、ESC キーが押されたら 0 以外を返します。</td></tr>
</table>

プリンタ制御コマンド

プリンタドライバが解釈する制御コマンドは、以下のとおりです。大文字はその記号、小文字はパラメータを表します。ESC は 01BH、CR は 0DH、LF は 0FH、FF は 0CH です。

表 3.37 プリンタ制御コマンド一覧

コマンド	意味
ESC R	フリンタのリセット プリンタをリセットします。
ESC V h v	グラフィック印刷の拡大率の設定 画面のグラフィック印刷を行なっときの拡大率を設定します。横拡大率は h で、縦拡大率は v で設定します。どちらも 1 バイトで指定し、1 を設定すると画面上の 1 ドットがプリンタ上での 1 ドットとなり、2 を設定すると画面上の 1 ドットがプリンタ上での 2 ドットとなります。
ESC F f	グラフィック印刷の左マージンの設定 画面のグラフィック印刷を行なうときの左マージンを設定します。f はワードで指定し、左端からのプリンタでのドット数となります。
ESC J j	紙送りの実行 紙送りを実行します。j はワードでプリンタでのドット数となります。
ESC ! n area1 area2 area3 ··· arean	
	画面のグラフィック印刷 画面のグラフィック印刷を行ないます。画面上のいくつかの範囲を一度に印刷できます。n はバイトでその後に続く画面の領域の個数を指定します。area1 から arean は AREA 構造体で印刷を行なう領域を指定します。画面上でパレットコードが 1 の画素がプリンタで黒として印刷されます。
ESC I pattern	アイコンパターンの印刷 12×12 ドットのアイコンを印刷します。pattern は、18 バイトのビットパターンデータそのものを指定します。
ESC K x y	テキスト印字の文字サイズの指定 テキスト印字での、文字のサイズを指定します。x はバイトで横のサイズを、y はバイトで縦ののサイズを指定します。文字サイズの初期値は 24×24 です。
ESC P p	テキスト印字の文字の横の間隔の指定 テキスト印字の文字の横の間隔を指定します。p はバイトです。文字ピッチの初期値は 1 です。

コマンド	意味
ESC X	アンダーラインテキスト印字の設定
	アンダーラインテキスト印字を設定します。これ以降、ESC Y が送られて解除されるまでのテキスト印字は、アンダーラインつきで行われます。アンダーライン印字は、1 行の印字が終了しても、解除されません。
ESC Y	アンダーラインテキスト印字の解除
	ESC X で設定されたアンダーライン印字を解除します。初期化時は、このモードになります。
ESC x	プロポーショナル印字の解除
	プロポーショナル印字を解除します。これ以降、テキスト印字は固定ピッチで印字されます。
ESC y	プロポーショナル印字の設定
	プロポーショナル印字を設定します。これ以降、テキスト印字はプロポーショナル印字されます。初期化時は、このモードになります。
CR	キャリッジリターン
	プリンタのバッファにあるデータを印刷し、プリンタヘッドを左端に戻します。
LF	ラインフィード
	プリンタのバッファにあるデータを印刷し、用紙を 1 行分送ります。
FF	フォームフィード
	プリンタのバッファにあるデータを印刷し、用紙を 1 ページ分送ります。
00H	印刷データの終わり
	印刷データの終了を意味します。プリンタドライバはこのデータを読むと、アプリケーションプログラムに返ります。

上記コマンド以外の文字が ESC の後に続くと、プリンタドライバはその ESC および次の文字を無視して、更にその次の文字から解釈を実行します。

上記プリンタ制御コマンド以外はテキスト印字として扱われます。漢字コードはシフト JIS コードです。テキスト印字で、漢字を印字できないプリンタでは、プリンタドライバが MSXView の持っている漢字フォントを展開して、ビットイメージで印字します。漢字プリンタでは、プリンタドライバがプリンタに合わせて、漢字コードを変換して送ります。

プリントマネージャの初期化

機能番号	311
書式	void _initprint(void)
戻り値	なし
解説	プリントマネージャを初期化します。このファンクションはシステムが自動的に実行するので、アプリケーションから呼び出す必要はありません。

プリンタドライバの変更

機能番号	312
書式	STATUS chpd(driver) char *driver [HL] プリンタドライバ名へのポインタ
戻り値	[A] OK 変更成功 ERROR 変更失敗
解説	プリンタドライバを driver で指定したものに変更します。

プリント情報の獲得

機能番号	313
書式	PINFO *_printinfo(void)
戻り値	[HL] PRINT構造体へのポインタ
解説	現在設定されているプリント情報の存在する PRINT構造体へのポインタを返します。

プリンタドライバの起動

機能番号　314

書　式

```
int   _pd(func)
WORD  func  [HL]  [illegible]番号
```

戻り値　[HL]　エラーが起こったら-1

解　説　プリンタドライバを起動し、func で指定されたプリントファンクションを実行します。

22章 オーバーレイの使い方

オーバーレイを使用するには、以下の2つの方法があります。

- _system() を用い、外部ファイルのオーバーレイモジュールを呼び出す方法
- _execute() を用いて、アプリケーションのファイルに結合されたオーバーレイモジュールを呼び出す方法

一般的に、前者はシステムで共通に使用するオーバーレイモジュールの呼び出しに使用し、後者はアプリケーションが個別に使用するオーバーレイモジュールの呼び出しに使用します。

前者の代表例としては、ファイル名を選択するための「FILEPACK.MV」や、フォントメニューを表示してフォントを選択させる「FONTMENU.MV」などがあります。

後者の_execute() を用いたオーバーレイモジュールは、個々のアプリケーションが使用する専用オーバーレイモジュールの場合に使用します。この方法では、オーバーレイモジュールも含めて、アプリケーションが1つのファイルになるため、アプリケーションの取り扱い（バックアップなど）が容易になるという利点があります。

オーバーレイモジュールは、0100Hからモジュールサイズだけ読み込まれます。オーバーレイモジュールの読み込みアドレスの指定はできません。したがって、オーバーレイモジュールがアクセスするデータやオーバーレイモジュールが利用するルーチン群は、メモリの高位アドレスに置かなければなりません。この際に、オーバーレイモジュールにより上書きされてしまうメモリの内容は、MSXViewが自動的に退避するので、アプリケーション側では退避に関する作業は何もする必要はありません。また、ディスク上の空き容量の許す限り、オーバーレイ呼び出しを多重に行なうこともできます。

オーバーレイモジュールが読み込まれると、無条件に0100Hがコールされます。したがって、オーバーレイモジュールの先頭には、必ずオーバーレイモジュールのmain関数へのジャンプ命令を挿入しておかなければなりません。さらに、MSXViewでは、このときに2つの引数を渡すことができます。1つめの引数をchar*、2つめの引数をintと仮定しているので呼び出されるモジュールは、main(char *parameter1, int parameter2) で受けるようにします。これ以外の型の引数が必要であれば、キャストして下さい。また、構造体などを引き渡すには、その構造体へのポインタを渡すのが一般的ですが、構造体のデータ格納アドレスが

オーバーレイモジュールによって上書きされてしまう可能性があるので、注意して下さい。このため、MSXViewでは、080H番地からの128バイト（MSX DOS2で使用されているデフォルトDMAバッファ）をMAILという名前で予約し、アプリケーションが自由に使用できるようにしています。

また、_execute()で、アプリケーションファイルに結合されたオーバーレイモジュールを呼び出すときは、アプリケーションファイルをカレントファイルにしておかなければなりません。このための手続きは、以下のようにします。

```
HANDLE  myfd;                        /* アプリケーションファイルのハンドル宣言 */
myfd = _fopen(MYNAME, NEW);          /* アプリケーションファイルをオープン */
_chfile(myfd);                       /* カレントファイルにする */
```

この手続きのあと、他のファイルをアクセスする際には、_pushfile()と_popfile()を使用し、_execute()を実行する際には、カレントファイルがアプリケーションファイルになっているように注意して下さい。

また、オーバーレイモジュールでは戻り値を返さないので、オーバーレイモジュールからの戻り値が必要なときは、_modulevalue()を使用して、戻り値を得て下さい。ここで返す型はWORDなので、構造体などを返すときは、MAILを使用することもできます。

詳しくは、16章「リソースマネージャ」の、_exequte()を参照して下さい。

第4部 MSX-MIDI

1章 MSX-MIDIとは

MSX-MIDI とは、これまでの MSX-MUSIC に MIDI (Musical Instrument DigitalInterface) 機能を追加したもので、拡張BASIC 命令で、MIDI を利用することができます。[illegible] MUSIC とは異なり、BIOS はありません。BASIC 以外で MSX-MIDI を使うプログラムは、I/O ポートから直接ハードウェアをアクセスします。

MSX-MIDI は本体に内蔵することも、外付けカートリッジにすることもできます。MSX-MIDIは、MSXturboR以降の MSX専用です。MSX、MSX_2、MSX_{2+}では使用できません。

MSX-MIDI は次のような構成になっています。

- MIDI インターフェイス

  8251　　　　　　MIDI データ通信用 IC

  8253 または 8254　ボーレートジェネレータおよびタイマー用IC

  これらのICに対しては、I/O ポートからアクセスします。

- MSX-MIDI ROM (16K バイト)

  本体内蔵の場合は、これまでの MSX-MUSIC と同じスロット (スロット 0-2、ページ 1) に配置します。

  外付けカートリッジの場合は、カートリッジ上に実装します。拡張BASIC命令を使うときは、本体内蔵のMSX-MUSICは使用せず、外付けカートリッジの MSX-MIDI を使うよう初期化されます。

2章
ハードウェア

MSX-MIDIは、内蔵と外付けによって、ハードウェア構成やアクセス方法が異なります。以下では、MSX-MIDIのハードウェア構成について説明します。

なお、タイマー用ICとして8253または8254を使用しますが、以下では「8253」と表記します。

2.1 ブロック図

MSX-MIDIのハードウェアは以下のような構成になっています。

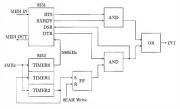



図 4.1 MSX-MIDIのブロック図

2.2 内蔵 MIDI インターフェイス

本体に内蔵された MSX-MIDI インターフェイスの I/O ポートは、次のように割り当てられています。

- 8251 インターフェイス　0E8H、0E9H 番地

		b7	b6	b5	b4	b3	b2	b1	b0
0E8H	(Read)	RXD7	RXD6	RXD5	RXD4	RXD3	RXD2	RXD1	RXD0
	(Write)	TXD7	TXD6	TXD5	TXD4	TXD3	TXD2	TXD1	TXD0

RXD7～RXD0　8251 受信データ
TXD7～TXD0　8251送信データ

		b7	b6	b5	b4	b3	b2	b1	b0
0E9H	(Read)	DSR	BRK	FE	OE	PE	EMPT	RRDY	TRDY

DSR	8253 タイマー割り込みフラグ	(1=割り込み発生)
BRK	8251 ブレークコード検出	(1=検出)
FE	8251 フレームエラーフラグ	(1=エラー発生)
OE	8251 オーバーランエラーフラグ	(1=エラー発生)
PE	8251 パリティエラーフラグ	(1=パリティエラー発生)
EMPT	8251 送信バッファステータス	(1=送信バッファ空)
RRDY	8251 受信バッファステータス	(1=データ有り)
TRDY	8251 送信ステータス	(1=送信可能)

		b7	b6	b5	b4	b3	b2	b1	b0
0E9H	(Write) モード	S2	S1	EP	PEN	L2	L1	B2	B1
	コマンド	EH	IR	RIE	ER	SBRK	RE	TIE	TEN

EH	通常 0 に設定します	
IR	通常 0 に設定します	
RIE	MIDI IN 割り込み許可	(1=許可、0=禁止)
ER	エラーリセット	(1=エラーフラグのリセット) (0=ノー・オペレーション)

SBRK	通常0に設定します	
RE	MIDI IN 受信イネーブル	(1=許可、0=禁止)
TIE	8253タイマー(カウンタ#2)割り込み許可	(1=許可、0=禁止)
TEN	MIDI OUT 送信イネーブル	(1=許可、0=禁止)

8251のコマンド・モードレジスタの書き込み回復時間は、最大16クロック(3.579545MHz)必要です。8251の初期化など、連続してコマンド・モードレジスタに書き込みをする際は、ウエイトを入れて下さい。
8251はI/Oポートの0E9Hに00H、00H、00H、40Hを書き込むと、リセットされます。モードに誤った値を設定すると、MIDIとして機能しなくなるので、リセット後は必ず特定の値を設定します。詳細は、添付のサンプルプログラム(「THRU.MAC」)を参照して下さい。

- 8253OUT2端子の信号のラッチ(0EAH、0EBH番地)

		b7	b6	b5	b4	b3	b2	b1	b0
0EAH	(Write)	—	—	—	—	—	—	—	—

0EAH 番地のデータリードは無効
0EBH 番地は0EAH 番地のイメージ

8253のカウンタ#2からの割り込みは、0EAH番地への任意のデータ書き込みによって解除されます。

- 8253 インターフェイス(0ECH~0EFH番地)

		b7	b6	b5	b4	b3	b2	b1	b0
0ECH	(R/W)	CT07	CT06	CT05	CT04	CT03	CT02	CT01	CT00
0EDH	(R/W)	CT17	CT16	CT15	CT14	CT13	CT12	CT11	CT10
0EEH	(R/W)	CT27	CT26	CT25	CT24	CT23	CT22	CT21	CT20
0EFH	(Read)	—	—	—	—	—	—	—	—
	(Write)	SC1	SC0	RW1	RW0	M2	M1	M0	BCD

CT07~CT00	カウンタ#0
CT17~CT10	カウンタ#1
CT27~CT20	カウンタ#2
SC1、SC0	カウンタ選択、コマンド選択
RW1、RW0	カウンタリードライトモード
M2、M1、M0	カウンタモード
BCD	バイナリ、BCD カウント選択

各カウンタの機能は次のようになっています。

- カウンタ#0

  8251のボーレートジェネレータとして使われます。CLK端子には4MHzのクロック信号が入力されています。8251に対しては、ボーレートクロックとして500KHzを送信する(8分周する)ように設定しなければなりません。モードは3(方形波N分周モード)で使用します。

- カウンタ#1

  汎用のカウンタとして使うことができます。CLK端子にはカウンタ#2の出力が入力されています。

- カウンタ#2

  CPUへの周期的な割り込みに使用されます(BASICでは5ミリ秒間隔の割り込みとして使用される)。通常、モード2(N分周モード)で使用します。OUT2端子がLOWになると、ラッチ回路を通してCPUに割り込みを発生します。CLK端子には4MHzのクロックが入力されます。

2.3 外付け MIDI インターフェイス

外付けのMSX-MIDIインターフェイスは、I/Oポートの0E2Hに値を設定することによって、I/Oポートのアドレスが変わります。

- MIDIインターフェイスの設定(0E2H、0EAH番地)

		b7	b6	b5	b4	b3	b2	b1	b0
0E2H	(Write)	EN	—	—	—	—	—	—	E8

EN	MIDIインターフェイスの使用許可、禁止 初期値は1。	(0=許可、1=禁止)
E8	8251アドレス設定 初期値は1。	(1=0E0H、0E1H番地) (0=0E8H、0E9H番地)

E8ビットを0に設定すると、外付けカートリッジのMIDIインターフェイスのI/Oポートは、0E2Hから0EAHに変わり、内蔵MIDIインターフェイスとコンパチブル

になります。また、8251 の I/O アドレスは、0E8H と 0E9H になります。

E8 ビットを 1 に設定すると、8251 インターフェイスは 0E0H、0E1H 番地になりカートリッジの I/O の 0ECH~0EFH へのアクセスは禁止されます。また、8253 のタイマー割り込みも禁止されます。

- 8251 インターフェイス (0E0H、0E1H 番地)

(E8 ビットが 1 の場合)

		b7	b6	b5	b4	b3	b2	b1	b0
0E0H	(Read)	RXD7	RXD6	RXD5	RXD4	RXD3	RXD2	RXD1	RXD0
	(Write)	TXD7	TXD6	TXD5	TXD4	TXD3	TXD2	TXD1	TXD0

RXD7~RXD0 8251 受信データ
TXD7~TXD0 8251 送信データ

		b7	b6	b5	b4	b3	b2	b1	b0
0E1H	(Read)	DSR	BRK	FE	OE	PE	EMPT	RRDY	TRDY

DSR	8253 タイマー割り込みフラグ	(1=割り込み発生)
BRK	8251 ブレークコード検出	(1=検出)
FE	8251 フレームエラーフラグ	(1=エラー発生)
OE	8251 オーバーランエラーフラグ	(1=エラー発生)
PE	8251 パリティエラーフラグ	(1=パリティエラー発生)
EMPT	8251 送信バッファステータス	(1=送信バッファ空)
RRDY	8251 受信バッファステータス	(1=データ有り)
TRDY	8251 送信ステータス	(1=送信可能)

		b7	b6	b5	b4	b3	b2	b1	b0
0E1H	(Write) モード	S2	S1	EP	PEN	L2	L1	B2	B1
	コマンド	EH	IR	RIE	ER	SBRK	RE	TIE	TEN

EH	通常0に設定します	
IR	通常0に設定します	
RIE	MIDI IN割り込み許可	(1=許可、0=禁止)
ER	エラーリセット	(1=エラーフラグのリセット) (0=ノー・オペレーション)
SBRK	通常0に設定します	
RE	MIDI IN受信イネーブル	(1=許可 0=禁止)
TIE	8253タイマー(カウンタ#2) 割り込み許可	(1=許可、0=禁止)
TEN	MIDI OUT送信イネーブル	(1=許可、0=禁止)

2.4 内蔵タイプと外付けタイプとの見分け方

MAIN ROMの002EH番地のビット0が1の場合は、MSX-MIDIは内蔵されています。

MAINROMのバージョン番号(【002DH】)が03H以上の機種では、外付けカートリッジでMSX-MIDIの機能を使うことができます。

内蔵タイプと外付けタイプとでは、フックが異なっています。この違いはアプリケーションを作成する上で重要です。フックについては、3.2「フック」と4章「アプリケーション開発の注意」を参照して下さい。

内蔵タイプと外付けタイプとを判別するには、MSX-MIDIのROMの4018H番地からの文字列を調べます。

表4.1 MSX-MIDIの判別用文字列

アドレス	内蔵	外付け
4018H	41H (A)	??H (?)
4019H	50H (P)	??H (?)
401AH	52H (R)	??H (?)
401BH	4CH (L)	??H (?)
401CH	4FH (O)	4DH (M)
401DH	50H (P)	49H (I)
401EH	4CH (L)	44H (D)
401FH	4CH (L)	49H (I)

外付けカートリッジでは、4018Hからの4バイトはメーカーごとに異なる任意のデータになります。401CHからのデータは「MIDI」となっています。

2.5 MIDI インターフェイスの有無の判別方法

MIDI インターフェイスの有無は、次のように判別します。

1. MAIN ROM の 002EH 番地のビット 0 が 1 の場合、MIDI インターフェイスは内蔵されています。
2. MAIN ROM のバージョン番号 (002DH) が 03H 以上の場合、401CH〜401FH に次の内容を持つスロットを探します。

```
DB      "MIDI"
```

   もしあれば、外付けカートリッジが実装されています。

3. 以上のことにあてはまらない場合、MIDI インターフェイスは存在しないので MIDI 機能は使用できません。
4. ROM のバージョン番号 (002DH) が 02H 以下の場合、MIDI インターフェイスは使用できません。

3章 割り込み

3.1 BASICでの割り込み

MSX-MUSICのための割り込みは、1/60秒（NTSC）または1/50秒（PAL）を使用してきましたが、MSX-MIDIの拡張BASICでは、8253からの5ミリ秒割り込みを使用します。

3.2 フック

MSXturbo R本体に内蔵されたMSX-MIDIのフックは次のようになっています。

表4.2 内蔵MSX-MIDIのフック

アドレス	名称	旧名称	内容
0FF75H	H.MDIN	H.OKNORM	MIDI IN 割り込み
0FF93H	H.MDTM	H.FRQINT	8253タイマー割り込み

外付けカートリッジの場合、これらのフックは使えないので、H.KEYIを使って下さい。使用法の詳細は、4章「アプリケーションの開発」で解説します。

表4.3 外付けMSX-MIDIのフック

アドレス	名称
0FD9AH	H.KEYI

4章 アプリケーションの開発

4.1 アプリケーション開発についての注意点

MSX-MIDI対応のアプリケーションプログラムを作成するときは、以下の点に注意して下さい。

1. フックは MSX-MIDI が本体に内蔵されているか、外付けされているかで異なります。フックを設定する際は、内蔵タイプか外付けタイプかを確認して下さい。
2. MIDI インターフェイス初期化後などに割り込みを許可するときは、すでに割り込みフラグがセットされている可能性があるので、割り込みフラグをリセットしなければなりません。
   割り込みフラグには、以下ものがあります。

表 4.4 MIDI インターフェイスの割り込みフラグ

割り込みの種類	割り込みの判別	割り込みの解除方法
タイマー	0E9H または 0E1H の bit7(DSR)	EAH への任意の書き込み
MIDI IN	0E9H または 0E1H の bit1(RRDY)	E8 H を IN 命令で読み込む

3. 外付けカートリッジでは、MIDI IN および 8253 タイマー以外の割り込みも H.KEYI に来ます。したがって、割り込み処理ルーチン内では、現在の割り込みがどういう割り込みなのか判別しなければなりません。
   MIDI IN 割り込みかどうかは、I/O ポート 0E9H 番地の bit1 で確認します。
   8253 タイマーからの割り込みかどうかは、I/O ポート 0E9H 番地の bit7 で確認します。

4、MIDI IN受信の割り込みは、最短320μ秒間隔で発生します。フックから割り込み処理ルーチンを呼ぶときにRST 30H命令を使うと、インタースロットコールの処理に時間がかかり、320μ秒間隔の受信に間に合わなくなります。
そのため、割り込みに関しては次のように設定して下さい。

- 割り込み処理ルーチンはページ3に置く。
- フックからはJP命令で割り込み処理ルーチンへジャンプする。

4.2 サンプルプログラム

MIDIインターフェイスとフックの設定・解除は、付属のサンプルプログラム（THRU.MAC）を参照して下さい。

5章 拡張BASIC

5.1 拡張 BASIC の概要

MSX MIDI には、各機能を簡単に使用できるように、MSX-MIDI拡張BASIC が用意されています。MSX-MIDI拡張BASICは、CALL MUSIC のように拡張ステートメントの形式になっています。CALL は、_ (アンダーバー) で代用できます。

MSX-MIDI拡張BASICでは、MIDI インターフェイスを通じて外部の MIDI 機器を使用することができます。そのため、コマンドが追加・変更されました。また、MML も MIDI に対応するため拡張・変更されました。

5.2 拡張 BASIC の解説

CALL MUSIC

機能

MSX-MIDI システムを初期化します。

書式

CALL MUSIC[([<モード>],[,[0] [,<PLAY 文第 1 文字列へのチャンネル数> [,<PLAY 文第 2 文字列へのチャンネル数> [,...[,<PLAY 文第 9 文字列へのチャンネル数,]]]]]]]]]])]

解説

内蔵 FM 音源 LSI の初期化、FM 音源のチャンネルをどのように使用するかの指定、MIDI インターフェイスの初期化を行います。CALLMUSIC 文により初期化するまでは、拡張 BASIC ステートメントを使うことはできません。

- <モード>
  指定するのは 0 か 1 で、以下のような意味があります。

指定	意味
0	リズム音を使用しない
1	リズム音を使用する

- チャンネル数

それぞれのモードで、使用できるチャンネル数は次のとおりです。

モード	リズム	最大チャンネル数
0	無し	9
1	有り	内蔵FM (6音+1リズム)
		MIDI (8音+1リズム)

モード1では、MIDIを使うときは使用できるチャンネル数が、8チャンネルまで増えました。

PLAY文の各文字列へのチャンネル数は、内蔵FM音源が使用し、その文字列がいくつのOPLLのチャンネルを占有するか指定します。0は指定できません。

OPLLのチャンネルは9までしかありません。また、モード1ではリズム音用に3チャンネル使います。したがって、モード0ではPLAY文の各文字列へのチャンネル数の合計は9以下、モード1では6以下でなければなりません。

MIDIの場合は、各文字列に対してMIDIチャンネルを1つ割り当てます。2以上の値を指定しても意味は変わりません。

FM音源のチャンネルは、チャンネルの小さい方から割り当てます。MIDIチャンネルは第1文字列から順に、1、2、3…と設定されます。これを変更するには、MMLの@Hnコマンドを使用します。

パラメータを1つ以上指定した場合、他のパラメータの省略時の値は0になります。

PLAY文の各文字列へのチャンネル数は、0に設定したり、省略したりすることはできませんが、一部例外があります。次の例を参照して下さい。

```
CALL MUSIC(0,0,0,5,0)
             ↑   ↑
0を設定してはいけない (Illegal function call になる)

CALL MUSIC(0,0,1,,2)
                ↑
省略してはいけない (Syntax error になる)
```

```
CALL MUSIC(1,0,1,1,1,1,1,1,0,0)
                         ↑ ↑
```

モード1の第7文字列、第8文字列に対してのみ0を設定できる。この時、PLAY文では次のようになる。

- PLAY #1の場合
  第1文字列～第8文字列はMIDIに割り当てる。
  第9文字列はMIDIのリズム音に割り当てる。
  第10文字列～第12文字列はPSGに割り当てる。
- PLAY #2の場合
  第1文字列～第6文字列は内蔵FM音源に割り当てる。
  第7文字列と第8文字列とは無視される。
  第9文字列は内蔵FM音源のリズム音に割り当てる。
  第10文字列～第12文字列はPSGに割り当てる。

パラメータなしで使われたときは、

```
CALLMUSIC(1,0,1,1,1)
```

と同じになります。

CALL MUSIC文を実行すると、システムの割り込みのフックがMSX-MIDIのシステムソフトウェアにリンクされるので、割り込み処理ルーチンのオーバーヘッドが増え、システムのスループットが低下します。また、CALL MUSIC文はワークエリア確保のために、内部でCLEAR文に相当することを行っているので、HIMEM（CLEAR文の第2パラメータに相当します）が807バイト小さく再設定され、変数はすべてクリアされます。

文　例　CALL MUSIC
デフォルトの設定をする。

CALL MUSIC(0,0,1,1,1,1,1,1,1,1,1)
1チャンネルずつPLAY文の文字列に割り当てる。

CALL MDR

機　能　リズム音用MMLで発音されるMIDIノート番号を設定します。

書　式　CALL MDR(<BのMIDIノート番号>, <SのMIDIノート番号>,<MのMIDIノート番号>, <CのMIDIノート番号>,<HのMIDIノート番号>)

解　説　リズム音用MMLで使用するB、S、M、C、Hコマンドに割り当てるMIDIノート番号を指定します。

MIDI音源の多くは、1つのチャンネルの各ノートにリズム音を割り当てる機能（リズム専用トラック）を持っています。PLAY文の各文字列は1つのチャンネルにしか対応していません。同時に複数音を発音するリズム音の場合は、このリズム専用トラックを使用するようにして下さい。
このリズム音の割り当ては、各MIDI機器によって異なります。お手持ちのMIDI機器のマニュアルを参考に割り当てて下さい。
パラメータは0~127を指定します。省略はできません。初期値はすべて0に設定されています。

文 例　CALLMDR(35,38,45,49,42)

PLAY

機 能　音楽をMMLにしたがって演奏します。

書 式　PLAY [#<モード>,]<文字列 1>[,<文字列 2> [,<文字列 3>...[,<文字列 13>]

解 説　PLAY文は音楽を演奏するもので、内蔵FM音源(9音)+PSG音源(3音)、またはMIDI機器(9音)+PSG音源(3音)の、最大12音まで同時発音が可能です。<文字列>に書かれたMMLにしたがって演奏します。
他の拡張命令と異なり、CALL文は必要ありません。
<モード>は0~3までの値をとり、PLAY文の音源や動作モードを次のように設定します。

- 0や省略されたときはPSGのみが音源となり、文字列は最大3つまでとなります。
- 1のとき、MIDI機器、PSG音源を使用できます。
  <文字列>との関係は始めから順に

  <MIDI機器用文字列 1>,...<MIDI機器用文字列 n>,
  <MIDI機器リズム音用文字列>,
  <PSG音源用文字列 1>,<PSG音源用文字列 2>,<PSG音源用文字列 3>

  となります。
- 2または3のとき、FM音源、PSG音源を使用できます（2のときと3のときとでは動作は同じ）。
  <文字列>との関係は初めから順に

  <内蔵FM音源用文字列 1>,...<内蔵FM音源用文字列 n>,
  <内蔵FM音源リズム音用文字列>
  <PSG音源用文字列 1>,<PSG音源用文字列 2>,<PSG音源用文字列 3>

となります。

n は CALL MUSIC 文で設定された MML の個数です。CALL MUSIC で リズム音を使用しないモードに設定した場合は、リズム音用文字列をカンマ (,) と共に省略しなければいけません。

文　例	PLAY #1,"CD","EF","GA"

MIDI 機器用 MML の仕様

ここでは、MIDI 機器用に追加および変更された MML を説明します。

表 4.5 MIDI 機器用に追加された MML

文字	意味	値の範囲
@Hn	使用する MIDI チャンネル番号を設定します。	1 ≦ n ≦ 16
@Cm,n	コントロールチェンジを出力します。	0 ≦ m ≦ 127
	m はコントロール番号、n はコントロール番号に対する値です。	0 ≦ n ≦ 127
@Sn	MIDI リアルタイムクロックに関する命令です。	
	n=0 FCH(STOP) を出力し、クロックを停止する	
	n=1 FAH(START) を出力し、クロック出力を開始する	
	n=2FBH(CONTINUE) を出力し、クロック出力を開始する	
	クロックのテンポは、PLAY 文の第 1 文字列と同じになります。	

表 4.6 MIDI 機器用に変更された MML

文字	意味	値の範囲
Ln	長さを設定します。	1 ≦ n ≦ 96
Rn	休符を設定します。	1 ≦ n ≦ 96
Wn	n で指定した長さだけ状態を継続します。	1 ≦ n ≦ 96
Vn	MIDI機器に対しては、8 倍した値をノートオン・ベロシティーとして出力します。	1 ≦ n ≦ 15
@Vn	ボリューム (コントロールチェンジ#7) を出力します。	1 ≦ n ≦ 127
@n	内蔵 FM 音源に対しては音色変更を出力しますが、	0 ≦ n ≦ 63
	MIDI 機器に対してはプログラムチェンジを出力します。	0 ≦ n ≦ 127
Zn	1 バイト MIDI データを出力します。	1 ≦ n ≦ 255

リズム音用MMLの仕様

ここでは、リズム音用に追加および変更されたMMLを説明します。

表4.7 リズム音用に追加されたMML

文字	意味
@n	MIDI機器に対してプログラムチェンジを出力します。内蔵FM音源では無視されます。

表4.8 リズム音用に変更されたMML

文字	意味
@An	MIDI機器に対しては、Vnと同様にベロシティーを設定します。

表 4.9 MML と各音源との対応一覧表

	文字	MIDI	FM	PSG
	Mn	−	−	○
	Sn			○
	Vn	○ [0〜15]	○	○
	Ln	○	○	○
	Qn	○	○	−
	On	○	○	○
	<	○	○	○
	>	○	○	○
	Tn	○	○	○
	Nn*1	○	○	○
	Rn	○	○	○
	A〜G	○	○	○
	+	○	○	○
	−	○	○	○
	#	○	○	○
		○	○	○
	=x;	○	○	○
	Xx;	○	○	○
	&	○	○	○
	{ }n	○	○	×
	@n	○ [0〜127]	○ [0〜63]	−
	@Vn	○ [0〜127]	○ [0〜127]	−
	@Wn	○	○	Rn と同等
★	@Hn	○ [1〜16]		−
★	@Cm、n	○ [0〜127]	−	−
	Yr,d	−	○	−
	Zn	○ [0〜255]	− (×)	−
★	@Sn	○ [0〜2]	−	−

1　n の値は MIDI ノートナンバーではありません。従来どおりの音階です。
★　MSX-MIDI で追加された命令です。
×　エラーになります。
−　無視されます。
[ ]　パラメ　ーの範囲を示します。
( )　旧版 (MSX−MUSIC) での対応を示します。

表 4.10 リズム音用 MML

	コマンド	MIDI	FM
	B*1	○	○
	S*1	○	○
	M*1	○	○
	C*1	○	○
	H*1	○	○
	n	○	○
	!	○	○
	Vn	○	○
	@An	○	○
	@Vn	○ [0~127]	○
	Tn	○	○
	Rn	○	○
	=x;	○	○
	Xx;	○	○
	Yr,d		○
★	@Hn	○ [1~16]	−
★	@Cm,n	○ [0~127]	−
★	@n	○ [0~127]	− (×)
	Zd	○	− (×)

1 MIDI機器の場合は、前もって発音されるノート番号を CALL MDR で定義しなければなりません。
★ MSX-MIDI で追加された命令です。
× エラーになります。
− 無視されます。
[ ] パラメータの範囲を示します。
( ) 旧版（MSX-MUSIC）での対応を示します。

5.3 MIDI 機器に対して無効なステートメント

以下のステートメントは、MIDI機器に対しては意味を持ちません。

CALL AUDREG
CALL PITCH
CALL TEMPER
CALL TRANSPOSE
CALL VOICE

MIDI機器の音色設定は CALL VOICE ではできません。MML の@n を使って設定します。

5.4 MIDI データフォーマット

MSX-MIDI 拡張 BASIC の MIDI データフォーマットは以下のようになっています。

5.4.1 送信

1. ノート送信

- ノート・オン

ステータス	1001nnnn(9nH)	n=0~15MIDI チャンネル番号
ノートナンバー	0kkkkkkk	k=24(C0)~119(B7)
ベロシティ	0vvvvvvv	v=8*I　I=0~15

ベロシティ値は Vn、@Anで指定された値を 8 倍にして出力します。

- ノート・オフ

ステータス	1001nnnn(9nH)	n=0~15 MIDI チャンネル番号
ノートナンバー	0kkkkkkk	k=24(C0)~119(B7)
ベロシティ	00000000(v=0)	ノートオフ

リズム音を複数音同時に発音または発音停止するときは、ランニングステータスで出力します。

2. コントロールチェンジ

- @Vv (ボリューム)

ステータス	1011nnnn(BnH)	n=0~15 MIDI チャンネル番号
コントロール番号	00000111	c=7
コントロール値	0vvvvvvv	v=0~127

- @Cc、v (コントロールチェンジ)

ステータス	1011nnnn(BnH)	n=0~15MIDI チャンネル番号
コントロール番号	0ccccccc	c=0~127
コントロール値	0vvvvvvv	v=0~127

3. プログラムチェンジ

- @p (プログラムチェンジ)

ステータス	1100nnnn(CnH)	n=0~15 MIDI チャンネル番号
プログラム番号	0ppppppp	p=0~127

4. システムリアルタイムメッセージ

- タイミングクロック
  ステータス　　11111000(F8H)
  タイミングクロックは、@S1 または@S2 でクロックが開始されていて、かつ PLAY 文が実行されているときにのみ送信されます。PLAY 文を終了すると、クロックは停止しますこの時、ストップ FCH は出力されません。
- @S1　スタート
  ステータス　　11111010(FAH)
  クロックの出力を開始します。
- @S2　コンティニュー
  ステータス　　11111011(FBH)
  クロックの出力を開始します。
- @S0　ストップ
  ステータス　　11111100(FCH)
  クロックの出力を停止します。

5.4.2 受信

拡張BASIC では受信は行いません。

表 4.11 MSX-MIDI 拡張BASIC MML MIDI インプリメンテーションチャート

ファンクション		送　信	備　考
ベーシック	電源ON時	1 - 9	
チャンネル	設定可能範囲	1 - 16	@Hn
	電源ON時	3	
モード	メッセージ	×	
	代用	*********	
ノート		○ 24 - 119	O1C - O8B
ナンバー	音域	*********	
ベロシティ	ノート・オン	○ 9nH , v=8×n	Vn , @An (n= 0 - 15) @An はリズム音のみ
	ノート・オフ	× 9nH , v=0	
アフター	キー別	×	
タッチ	チャンネル別	×	
ピッチ・ベンド		×	
コントロール チェンジ		○ (7) ○ (0 - 127)	@Vn @Cm , n
プログラム		○	@n
チェンジ	設定可能範囲	*********	
エクスクルーシブ		×	
	ソング・ポジション	×	
コモン	ソング・セレクト	×	
	チューン	×	
リアル	クロック	○	
タイム	コマンド	○	@Sn
	ローカルON/OFF	×	
その他	オール・ノート・オフ	○	[CTRL] + [STOP]
	アクティブ・センシング	×	
	リセット	×	
備考		Zn : 1バイトデータの送信	

モード1：オムニ・オン、ポリ　モード2：オムニ・オン、モノ　○：あり
モード3：オムニ・オフ、ポリ　モード4：オムニ・オフ、モノ　×：なし
********* は送信では意味を持たない

APPENDIX

A R800インストラクション表

この表は、命令の種類ごとに分類して、R800のインストラクションをまとめたものです。表中のニーモニックは各命令の名前を現わし、命令動作でその動作内容を簡単に示しています。

命令動作の欄で←とあるのは、右側の内容を左側に代入することを、カッコでくくられたものは、くくられたレジスタなどで示されるメモリの内容を、それぞれ意味しています。例えば

```
r ← [.hl]
```

とあるのは、.hl レジスタで示されるアドレスのメモリの内容を、8ビットレジスタに代入するということです。ただし、入出力命令の [n] と [.c] は、対応する入出力ポートの番号のことです。

フラグの欄は各フラグの動作を、オペコードはそれぞれの命令に対するマシン語コードを、2進数と16進数で記したものです。その右側の B と C は、各命令の長さ(バイト数)と、命令を実行するのに要するクロック数を、それぞれ表しています。

このほか、インストラクション表に出てくる略号に関しては、次の凡例を参照して下さい。

表に記載されたニーモニックは Z80 のものと違っていますが、R800 で追加された乗算命令や、Z80 で正式に動作が保証されていなかった命令以外は、Z80 と命令動作はすべて同じです。

凡例

a(7)	レジスター a の最上位ビット
a(4-7)	レジスター a のビット 4-7
,	動作の区切り
de:hl	上位 16 ビットが de、下位 16 ビットが hl に入る、32 ビット整数
[ix+d]	ix に 8 ビットの符号つき変位を足した値が示すアドレス
C	キャリーフラグ
Z	ゼロフラグ
P/V	パリティー・オーバーフローフラグ
S	サインフラグ
N	減算フラグ
H	ハーフキャリーフラグ
•	フラグは変化しない
I	フラグは実行結果により変化する
0	フラグは 0
1	フラグは 1
?	不定になる
V	オーバーフローフラグとして使われる
P	パリティーフラグとして使われる
**	割り込みフリップフロップの値が入る
r,r'	8 ビットレジスター、a, b, c, d, e, h, l
u,u'	8 ビットレジスター、a, b, c, d, e, ixh, ixl
v,v'	8 ビットレジスター、a, b, c, d, e, iyh, iyl
p	8 ビットレジスター、ixh, ixl
q	8 ビットレジスター、iyh, iyl
ss	16 ビットレジスター、bc, de, hl, sp
pp	16 ビットレジスター、bc, de, ix, sp
rr	16 ビットレジスター、bc, de, iy, sp
qq	16 ビットレジスター、bc, de, hl, af
e	short br 系の命令の飛び先アドレ[illegible]即値 (+127～-128)
k	brk 命令の飛び先アドレス、00h,08h,10h,18h,20h,28h,30h,38h
nn	16 ビットの即値、もしくは絶対アドレス
n	8 ビットの即値
b	ビット演算命令の第何ビットかを示す値
NOT	ビットを反転する
∨	ビットの OR をとる
⊻	ビットの XOR をとる
∧	ビットの AND をとる
tmp	一時的に値を待避する
B	命令のバイト数
C	命令の実行に必要な最小クロック数

分岐命令、コール命令でクロック数が 2 つ書いてあるものは、上が条件が成立しないとき下が条件が成立したときを意味します。

また、入出力命令でクロック数が 2 つ書いてあるものは、上がまだ転送が終わらないとき下が転送が終わったときをそれぞれ意味します。

ここに記す命令表のクロック数は、SYSCLK 換算で XTAL の発振周波数の 4 分の 1 です。また、ノーウエイトで実行したときの値で、DRAM 上で実行したときはページブレークやリフレッシュにより、自動的にウエイトが挿入されます。

A.1 8 ビット移動命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
ld r,r'	r←r'	• • • • • •	01 r r'		1	1
ld r,n	r←n	• • • • • •	00 r 110 ← n →		2	2
ld r,[.hl]	r←[.hl]	• • • • • •	01 r 110		1	2
ld r,[.ix+d]	r←[.ix+d]	• • • • • •	11011101 01 r 110 ← d →	DD	3	5
ld r,[.iy+d]	r←[.iy+d]	• • • • • •	11111101 01 r 110 ← d →	FD	3	5
ld [.hl],r	[.hl]←r	• • • • • •	01110 r		1	2
ld [.ix+d],r	[.ix+d]←r	• • • • • •	11011101 01110 r ← d →	DD	3	5
ld [.iy+d],r	[.iy+d]←r	• • • • • •	11111101 01110 r ← d →	FD	3	5
ld u,u'	u←u'	• • • • • •	11011101 01 u u'	DD	2	2
ld v,v'	v←v'	• • • • • •	11111101 01 v v'	FD	2	2
ld u,n	u←n	• • • • • •	11011101 00 u 110 ← n →	DD	3	3
ld v,n	v←n	• • • • • •	11111101 00 v 110 ← n →	FD	3	3
ld [.hl],n	[.hl]←n	• • • • • •	00110110 ← n →	36	2	3
ld [.ix+d],n	[.ix+d]←n	• • • • • •	11011101 00110110 ← d → ← n →	DD 36	4	5
ld [.iy+d],n	[.iy+d]←n	• • • • • •	11111101 00110110 ← d → ← n →	FD 36	4	5

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
ld .a,.i	.a←.i	↕ ↕ 0 IFF 0 •	11101101 01010111	ED 57	2	2
ld .a,.r	.a←.r	↕ ↕ 0 IFF 0 •	11101101 01011111	ED 5F	2	2
ld .i,.a	.i←.a	• • • • • •	11101101 01000111	ED 47	2	2
ld .r,.a	.r←.a	• • • • • •	11101101 01001111	ED 4F	2	2
ld .a,[.bc]	.a←[.bc]	• • • • • •	00001010	0A	1	2
ld .a,[.de]	.a←[.de]	• • • • • •	00011010	1A	1	2
ld .a,[nn]	.a←[nn]	• • • • • •	00111010 ← nn_l → ← nn_h →	3A	3	4
ld [.bc],.a	[.bc]←.a	• • • • • •	00000010	02	1	2
ld [.de],.a	[.de]←.a	• • • • • •	00010010	12	1	2
ld [nn],.a	[nn]←.a	• • • • • •	00110010 ← nn_l → ← nn_h →	32	3	4

	000	001	010	011	100	101	110	111
r	.b	.c	.d	.e	.h	.l		.a
u	.b	.c	.d	.e	.ixh	.ixl		.a
v	.b	.c	.d	.e	.iyh	.iyl		.a

A.2 16 ビット移動命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
ld ss,nn	ss←nn	• • • • • •	00 ss 0001 ← nn_l → ← nn_h →		3	3
ld .ix,nn	.ix←nn	• • • • • •	11011101 00100001 ← nn_l → ← nn_h →	DD 21	4	4
ld .iy,nn	.iy←nn	• • • • • •	11111101 00100001 ← nn_l → ← nn_h →	FD 21	4	4
ld .sp,.hl	.sp←.hl	• • • • • •	11111001	F9	1	1
ld .sp,.ix	.sp←.ix	• • • • • •	11011101 11111001	DD F9	2	2
ld .sp,.iy	.sp←.iy	• • • • • •	11111101 11111001	FD F9	2	2

ニーモニック	命令動作	flags	オペコード			
		S Z H P/V N C	76 543 210	Hex	B	C
ld ss,[nn]	ss_h←[nn+1] ss_l←[nn]	• • • • • •	11 101 101 01 ss 1011 ← nn_l → ← nn_h →	ED	4	6
ld .hl,[nn]	.h←[nn+1] .l←[nn]	• • • • • •	00 101 010 ← nn_l → ← nn_h →	2A	3	5
ld .ix,[nn]	.ixh←[nn+1] .ixl←[nn]	• • • • • •	11 011 101 00 101 010 ← nn_l → ← nn_h →	DD 2A	4	6
ld .iy,[nn]	.iyh←[nn+1] .iyl←[nn]	• • • • • •	11 111 101 00 101 010 ← nn_l → ← nn_h →	FD 2A	4	6
ld [nn],ss	[nn+1]←ss_h [nn]←ss_l	• • • • • •	11 101 101 01 ss 0011 ← nn_l → ← nn_h →	ED	4	6
ld [nn],.hl	[nn+1]←.h [nn]←.l	• • • • • •	00 100 010 ← nn_l → ← nn_h →	22	3	5
ld [nn],.ix	[nn+1]←.ixh [nn]←.ixl	• • • • • •	11 011 101 00 100 010 ← nn_l → ← nn_h →	DD 22	4	6
ld [nn],.iy	[nn+1]←.iyh [nn]←.iyl	• • • • • •	11 111 101 00 100 010 ← nn_l → ← nn_h →	FD 22	4	6

	00	01	10	11
ss	.bc	.de	.hl	.sp

A.3 交換命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	 Hex	 B	 C
xch .de,.hl	.de↔.hl	• • • • • •	11101011	EB	1	1
xch .af,.af'	.af↔.af'	↕ ↕ ↕ ↕ ↕ ↕	00001000	08	1	1
xch [.sp],.hl	.l↔[.sp];.h↔[.sp+1]	• • • • • •	11100011	E3	1	5
xch [.sp],.ix	.ixl↔[.sp] .ixh↔[.sp+1]	• • • • • •	11011101 11100011	DD E3	2	6
xch [.sp],.iy	.iyl↔[.sp] .iyh↔[.sp+1]	• • • • • •	11111101 11100011	FD E3	2	6
xchx	.bc↔.bc';.de↔.de';.hl↔.hl'	• • • • • •	11011001	D9	1	1

A.4 スタック操作命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	 Hex	 B	 C
push qq	[.sp−2]←qq$_l$;[.sp−1]←qq$_h$ .sp←.sp−2	• • • • • •	11qq0101		1	4
push .ix	[.sp−2]←.ixl;[.sp−1]←.ixh .sp←.sp−2	• • • • • •	11011101 11100101	DD E5	2	5
push .iy	[.sp−2]←.iyl;[.sp−1]←.iyh .sp←.sp−2	• • • • • •	11111101 11100101	FD E5	2	5
pop qq	qq$_l$←[.sp];qq$_h$←[.sp+1] .sp←.sp+2	• • • • • •	11qq0001		1	3
pop .ix	.ixl←[.sp];.ixh←[.sp+1] .sp←.sp+2	• • • • • •	11011101 11100001	DD E1	2	4
pop .iy	.iyl←[.sp];.iyh←[.sp+1] .sp←.sp+2	• • • • • •	11111101 11100001	FD E1	2	4

	00	01	10	11
qq	.bc	.de	.hl	.af

pop .af のときは flags はすべて変化する

A.5 ブロック転送命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
move [.hl++],[.de++]	[.de]←[.hl];.de←.de+1 .hl←.hl+1;.bc←.bc−1	• • 0 ↕ 0 • *1	11101101 10100000	ED A0	2	4
move [.hl−−],[.de−−]	[.de]←[.hl];.de←.de−1 .hl←.hl−1;.bc←.bc−1	• • 0 ↕ 0 • *1	11101101 10101000	ED A8	2	4
movem [.hl++],[.de++]	repeat;[.de]←[.hl];.de←.de+1 .hl←.hl+1;.bc←.bc−1;until .bc=0	• • 0 0 0 •	11101101 10110000	ED B0	2	4
movem [.hl−−],[.de−−]	repeat;[.de]←[.hl];.de←.de−1 .hl←.hl−1;.bc←.bc−1;until .bc=0	• • 0 0 0 •	11101101 10111000	ED B8	2	4

*1.bc−1=0 のとき 0、その他 1

A.6 ブロックサーチ命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
cmp .a,[.hl++]	.a−[.hl];.hl←.hl+1 .bc←.bc−1	↕ ↕ ↕ ↕ 1 • *2 *1	11101101 10100001	ED A1	2	4
cmp .a,[.hl−−]	.a−[.hl];.hl←.hl−1 .bc←.bc−1	↕ ↕ ↕ ↕ 1 • *2 *1	11101101 10101001	ED A9	2	4
cmpm .a,[.hl++]	repeat;.a−[.hl];.hl←.hl+1 .bc←.bc−1;until .bc=0 or .a=[.hl]	↕ ↕ ↕ ↕ 1 • *2 *1	11101101 10110001	ED B1	2	5
cmpm .a,[.hl−−]	repeat;.a−[.hl];.hl←.hl−1 .bc←.bc−1;until .bc=0 or .a=[.hl]	↕ ↕ ↕ ↕ 1 • *2 *1	11101101 10111001	ED B9	2	5

*1.bc−1=0 のとき 0、その他 1

*2.a=[.hl] のとき 1、その他 0

A.7 乗算命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
mulub .a,r	.hl←.a*r	0 ↕ • 0 • ↕	11101101 11 r 001	ED	2	14
muluw .hl,ss	.de:.hl←.hl*ss	0 ↕ • 0 • ↕	11101101 11 ss 0011	ED	2	36

mulub では r が.b,.c,.d,.e のとき以外は動作が保証されない

muluw では ss が.bc,.sp のとき以外は動作が保証されない

A.8 加算命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
add .a,r	.a←.a+r	↕ ↕ ↕ V 0 ↕	10000 r		1	1
add .a,p	.a←.a+p	↕ ↕ ↕ V 0 ↕	11011101 10000 p	DD	2	2
add .a,q	.a←.a+q	↕ ↕ ↕ V 0 ↕	11111101 10000 q	FD	2	2
add .a,[.hl]	.a←.a+[.hl]	↕ ↕ ↕ V 0 ↕	10000110	86	1	2
add .a,[.ix+d]	.a←.a+[.ix+d]	↕ ↕ ↕ V 0 ↕	11011101 10000110 ← d →	DD 86	3	5
add .a,[.iy+d]	.a←.a+[.iy+d]	↕ ↕ ↕ V 0 ↕	11111101 10000110 ← d →	FD 86	3	5
add .a,n	.a←.a+n	↕ ↕ ↕ V 0 ↕	11000110 ← n →	C6	2	2
addc .a,r	.a←.a+r+C	↕ ↕ ↕ V 0 ↕	10001 r		1	1
addc .a,p	.a←.a+p+C	↕ ↕ ↕ V 0 ↕	11011101 10001 p	DD	2	2
addc .a,q	.a←.a+q+C	↕ ↕ ↕ V 0 ↕	11111101 10001 q	FD	2	2
addc .a,[.hl]	.a←.a+[.hl]+C	↕ ↕ ↕ V 0 ↕	10001110	8E	1	2
addc .a,[.ix+d]	.a←.a+[.ix+d]+C	↕ ↕ ↕ V 0 ↕	11011101 10001110 ← d →	DD 8E	3	5
addc .a,[.iy+d]	.a←.a+[.iy+d]+C	↕ ↕ ↕ V 0 ↕	11111101 10001110 ← d →	FD 8E	3	5
addc .a,n	.a←.a+n+C	↕ ↕ ↕ V 0 ↕	11001110 ← n →	CE	2	2
addc .hl,ss	.hl←.hl+ss+C	↕ ↕ ? V 0 ↕	11101101 01 ss 1010	ED	2	2
add .hl,ss	.hl←.hl+ss	• • ? • 0 ↕	00 ss 1001		1	1
add .ix,pp	.ix←.ix+pp	• • ? • 0 ↕	11011101 00 pp 1001	DD	2	2
add .iy,rr	.iy←.iy+rr	• • ? • 0 ↕	11111101 00 rr 1001	FD	2	2

ニーモニック	命令動作	flags	オペコード		B	C
		S Z H P/V N C	76543210	Hex		
inc r	r←r+1	↕ ↕ ↕ V 0 •	00 r 100		1	1
inc p	p←p+1	↕ ↕ ↕ V 0 •	11011101 00 p 100	DD	2	2
inc q	q←q+1	↕ ↕ ↕ V 0 •	11111101 00 q 100	FD	2	2
inc [.hl]	[.hl]←[.hl]+1	↕ ↕ ↕ V 0 •	00110100	34	1	4
inc [.ix+d]	[.ix+d]←[.ix+d]+1	↕ ↕ ↕ V 0 •	11011101 00110100 ← d →	DD 34	3	7
inc [.iy+d]	[.iy+d]←[.iy+d]+1	↕ ↕ ↕ V 0 •	11111101 00110100 ← d →	FD 34	3	7
inc ss	ss←ss+1	• • • • • •	00 ss 0011		1	1
inc .ix	.ix←.ix+1	• • • • • •	11011101 00100011	DD 23	2	2
inc .iy	.iy←.iy+1	• • • • • •	11111101 00100011	FD 23	2	2

	00	01	10	11
ss	.bc	.de	.hl	.sp
pp	.bc	.de	.ix	.sp
rr	.bc	.de	.iy	.sp

	000	001	010	011	100	101	110	111
p					.ixh	.ixl		
q					.iyh	.iyl		

A.9 減算命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
sub .a,r	.a←.a−r	↕ ↕ ↕ V 1 ↕	10010 r		1	1
sub .a,p	.a←.a−p	↕ ↕ ↕ V 1 ↕	11011101 10010 p	DD	2	2
sub .a,q	.a←.a−q	↕ ↕ ↕ V 1 ↕	11111101 10010 q	FD	2	2
sub .a,[.hl]	.a←.a−[.hl]	↕ ↕ ↕ V 1 ↕	10010110	96	1	2
sub .a,[.ix+d]	.a←.a−[.ix+d]	↕ ↕ ↕ V 1 ↕	11011101 10010110 ← d →	DD 96	3	5
sub .a,[.iy+d]	.a←.a−[.iy+d]	↕ ↕ ↕ V 1 ↕	11111101 10010110 ← d →	FD 96	3	5
sub .a,n	.a←.a−n	↕ ↕ ↕ V 1 ↕	11010110 ← n →	D6	2	2
subc.a,r	.a←.a−r−c	↕ ↕ ↕ V 1 ↕	10011 r		1	1
subc.a,p	.a←.a−p−c	↕ ↕ ↕ V 1 ↕	11011101 10011 p	DD	2	2
subc.a,q	.a←.a−q−c	↕ ↕ ↕ V 1 ↕	11111101 10011 q	FD	2	2
subc.a,[.hl]	.a←.a−[.hl]−c	↕ ↕ ↕ V 1 ↕	10011110	9E	1	2
subc.a,[.ix+d]	.a←.a−[.ix+d]−c	↕ ↕ ↕ V 1 ↕	11011101 10011110 ← d →	DD 9E	3	5
subc.a,[.iy+d]	.a←.a−[.iy+d]−c	↕ ↕ ↕ V 1 ↕	11111101 10011110 ← d →	FD 9E	3	5
subc.a,n	.a←.a−n−c	↕ ↕ ↕ V 1 ↕	11011110 ← n →	DE	2	2
subc.hl,ss	.hl←.hl−ss−c	↕ ↕ ? V 1 ↕	11101101 01ss0010	ED	2	2

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
dec r	r←r−1	↕ ↕ ↕ V 1 •	00 r 101		1	1
dec p	p←p−1	↕ ↕ ↕ V 1 •	11011101 00 p 101	DD	2	2
dec q	q←q−1	↕ ↕ ↕ V 1 •	11111101 00 q 101	FD	2	2
dec [.hl]	[.hl]←[.hl]−1	↕ ↕ ↕ V 1 •	00110101	35	1	4
dec [.ix+d]	[.ix+d]←[.ix+d]−1	↕ ↕ ↕ V 1 •	11011101 00110101 ← d →	DD 35	3	7
dec [.iy+d]	[.iy+d]←[.iy+d]−1	↕ ↕ ↕ V 1 •	11111101 00110101 ← d →	FD 35	3	7
dec ss	ss←ss−1	• • • • • •	00 ss 1011		1	1
dec .ix	.ix←.ix−1	• • • • • •	11011101 00101011	DD 2B	2	2
dec .iy	.iy←.iy−1	• • • • • •	11111101 00101011	FD 2B	2	2

A.10 比較命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
cmp .a,r	.a−r	↕ ↕ ↕ V 1 ↕	10111 r		1	1
cmp .a,p	.a−p	↕ ↕ ↕ V 1 ↕	11011101 10111 p	DD	2	2
cmp .a,q	.a−q	↕ ↕ ↕ V 1 ↕	11111101 10111 q	FD	2	2
cmp .a,[.hl]	.a−[.hl]	↕ ↕ ↕ V 1 ↕	10111110	BE	1	2
cmp .a,[.ix+d]	.a−[.ix+d]	↕ ↕ ↕ V 1 ↕	11011101 10111110 ← d →	DD BE	3	5
cmp .a,[.iy+d]	.a−[.iy+d]	↕ ↕ ↕ V 1 ↕	11111101 10111110 ← d →	FD BE	3	5
cmp .a,n	.a−n	↕ ↕ ↕ V 1 ↕	11111110 ← n →	FE	2	2

A.11 論理演算命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	 Hex	 B	 C
and .a,r	.a←.a∧r	↕ ↕ 1 P 0 0	10100 r		1	1
and .a,p	.a←.a∧p	↕ ↕ 1 P 0 0	11011101 10100 p	DD	2	2
and .a,q	.a←.a∧q	↕ ↕ 1 P 0 0	11111101 10100 q	FD	2	2
and .a,[.hl]	.a←.a∧[.hl]	↕ ↕ 1 P 0 0	10100110	A6	1	2
and .a,[.ix+d]	.a←.a∧[.ix+d]	↕ ↕ 1 P 0 0	11011101 10100110 ← d →	DD A6	3	5
and .a,[.iy+d]	.a←.a∧[.iy+d]	↕ ↕ 1 P 0 0	11111101 10100110 ← d →	FD A6	3	5
and .a,n	.a←.a∧n	↕ ↕ 1 P 0 0	11100110 ← n →	E6	2	2
or .a,r	.a←.a∨r	↕ ↕ 0 P 0 0	10110 r		1	1
or .a,p	.a←.a∨p	↕ ↕ 0 P 0 0	11011101 10110 p	DD	2	2
or .a,q	.a←.a∨q	↕ ↕ 0 P 0 0	11111101 10110 q	FD	2	2
or .a,[.hl]	.a←.a∨[.hl]	↕ ↕ 0 P 0 0	10110110	B6	1	2
or .a,[.ix+d]	.a←.a∨[.ix+d]	↕ ↕ 0 P 0 0	11011101 10110110 ← d →	DD B6	3	5
or .a,[.iy+d]	.a←.a∨[.iy+d]	↕ ↕ 0 P 0 0	11111101 10110110 ← d →	FD B6	3	5
or .a,n	.a←.a∨n	↕ ↕ 0 P 0 0	11110110 ← n →	F6	2	2
xor .a,r	.a←.a⊻r	↕ ↕ 0 P 0 0	10101 r		1	1
xor .a,p	.a←.a⊻p	↕ ↕ 0 P 0 0	11011101 10101 p	DD	2	2
xor .a,q	.a←.a⊻q	↕ ↕ 0 P 0 0	11111101 10101 q	FD	2	2
xor .a,[.hl]	.a←.a⊻[.hl]	↕ ↕ 0 P 0 0	10101110	AE	1	2
xor .a,[.ix+d]	.a←.a⊻[.ix+d]	↕ ↕ 0 P 0 0	11011101 10101110 ← d →	DD AE	3	5
xor .a,[.iy+d]	.a←.a⊻[.iy+d]	↕ ↕ 0 P 0 0	11111101 10101110 ← d →	FD AE	3	5
xor .a,n	.a←.a⊻n	↕ ↕ 0 P 0 0	11101110 ← n →	EE	2	2

A.12　ビット操作命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
bit b,r	z←NOT r$_{(b)}$	? ↕ 1 ? 0 •	11001011 01 b r	CB	2	2
bit b,[.hl]	z←NOT [.hl]$_{(b)}$	? ↕ 1 ? 0 •	11001011 01 b 110	CB	2	3
bit b,[.ix+d]	z←NOT [.ix+d]$_{(b)}$	? ↕ 1 ? 0 •	11011101 11001011 ← d → 01 b 110	DD CB	4	5
bit b,[.iy+d]	z←NOT [.iy+d]$_{(b)}$	? ↕ 1 ? 0 •	11111101 11001011 ← d → 01 b 110	FD CB	4	5
set b,r	r$_{(b)}$←1	• • • • • •	11001011 11 b r	CB	2	2
set b,[.hl]	[.hl]$_{(b)}$←1	• • • • • •	11001011 11 b 110	CB	2	5
set b,[.ix+d]	[.ix+d]$_{(b)}$←1	• • • • • •	11011101 11001011 ← d → 11 b 110	DD CB	4	7
set b,[.iy+d]	[.iy+d]$_{(b)}$←1	• • • • • •	11111101 11001011 ← d → 11 b 110	FD CB	4	7
clr b,r	r$_{(b)}$←0	• • • • • •	11001011 10 b r	CB	2	2
clr b,[.hl]	[.hl]$_{(b)}$←0	• • • • • •	11001011 10 b 110	CB	2	5
clr b,[.ix+d]	[.ix+d]$_{(b)}$←0	• • • • • •	11011101 11001011 ← d → 10 b 110	DD CB	4	7
clr b,[.iy+d]	[.iy+d]$_{(b)}$←0	• • • • • •	11111101 11001011 ← d → 10 b 110	FD CB	4	7

A.13　ローテイト命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
rola	C←.a$_{(7)}$;.a←.a*2;.a$_{(0)}$←C	• • 0 • 0 ↕	00000111	07	1	1
rora	C←.a$_{(0)}$;.a←.a/2;.a$_{(7)}$←C	• • 0 • 0 ↕	00001111	0F	1	1
rolca	tmp←C;C←.a$_{(7)}$;.a←.a*2;.a$_{(0)}$←tmp	• • 0 • 0 ↕	00010111	17	1	1
rorca	tmp←C;C←.a$_{(0)}$;.a←.a/2;.a$_{(7)}$←tmp	• • 0 • 0 ↕	00011111	1F	1	1
rol　r	C←r$_{(7)}$ r←r*2;r$_{(0)}$←C	↕ ↕ 0 P 0 ↕	11001011 00000 r	CB	2	2
rol　[.hl]	C←[.hl]$_{(7)}$ [.hl]←[.hl]*2;[.hl]$_{(0)}$←C	↕ ↕ 0 P 0 ↕	11001011 00000110	CB 06	2	5
rol　[.ix+d]	C←[.ix+d]$_{(7)}$ [.ix+d]←[.ix+d]*2 [.ix+d]$_{(0)}$←C	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00000110	DD CB 06	4	7
rol　[.iy+d]	C←[.iy+d]$_{(7)}$ [.iy+d]←[.iy+d]*2 [.iy+d]$_{(0)}$←C	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00000110	FD CB 06	4	7
ror　r	C←r$_{(0)}$ r←r/2;r$_{(7)}$←C	↕ ↕ 0 P 0 ↕	11001011 00001 r	CB	2	2
ror　[.hl]	C←[.hl]$_{(0)}$ [.hl]←[.hl]/2;[.hl]$_{(7)}$←C	↕ ↕ 0 P 0 ↕	11001011 00001110	CB 0E	2	5
ror　[.ix+d]	C←[.ix+d]$_{(0)}$ [.ix+d]←[.ix+d]/2 [.ix+d]$_{(7)}$←C	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00001110	DD CB 0E	4	7
ror　[.iy+d]	C←[.iy+d]$_{(0)}$ [.iy+d]←[.iy+d]/2 [.iy+d]$_{(7)}$←C	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00001110	FD CB 0E	4	7

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	 Hex	B	C
rolc r	tmp←C;C←r$_{(7)}$ r←r*2;r$_{(0)}$←tmp	↕ ↕ 0 P 0 ↕	11001011 00010 r	CB	2	2
rolc [.hl]	tmp←C;C←[.hl]$_{(7)}$ [.hl]←[.hl]*2;[.hl]$_{(0)}$←tmp	↕ ↕ 0 P 0 ↕	11001011 00010110	CB 16	2	5
rolc [.ix+d]	tmp←C C←[.ix+d]$_{(7)}$ [.ix+d]←[.ix+d]*2 [.ix+d]$_{(0)}$←tmp	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00010110	DD CB 16	4	7
rolc [.iy+d]	tmp←C C←[.iy+d]$_{(7)}$ [.iy+d]←[.iy+d]*2 [.iy+d]$_{(0)}$←tmp	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00010110	FD CB 16	4	7
rorc r	tmp←C;C←r$_{(0)}$ r←r/2;r$_{(7)}$←tmp	↕ ↕ 0 P 0 ↕	11001011 00011 r	CB	2	2
rorc [.hl]	tmp←C;C←[.hl]$_{(0)}$ [.hl]←[.hl]/2;[.hl]$_{(7)}$←tmp	↕ ↕ 0 P 0 ↕	11001011 00011110	CB 1E	2	5
rorc [.ix+d]	tmp←C C←[.ix+d]$_{(0)}$ [.ix+d]←[.ix+d]/2 [.ix+d]$_{(7)}$←tmp	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00011110	DD CB 1E	4	7
rorc [.iy+d]	tmp←C C←[.iy+d]$_{(0)}$ [.iy+d]←[.iy+d]/2 [.iy+d]$_{(7)}$←tmp	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00011110	FD CB 1E	4	7
rol4 [.hl]	tmp←.a$_{(0\ 3)}$;.a$_{(0\ 3)}$←[.hl]$_{(4\ 7)}$ [.hl]$_{(4\ 7)}$←[.hl]$_{(0\ 3)}$;[.hl]$_{(0\ 3)}$←tmp	↕ ↕ 0 P 0 •	11101101 11101111	ED 6F	2	5
ror4 [.hl]	tmp←.a$_{(0\ 3)}$;.a$_{(0\ 3)}$←[.hl]$_{(0\ 3)}$ [.hl]$_{(0\ 3)}$←[.hl]$_{(4\ 7)}$;[.hl]$_{(4\ 7)}$←tmp	↕ ↕ 0 P 0 •	11101101 11100111	ED 67	2	5

A.14 シフト命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
shl r shla	C←r(7) r←r*2	↕ ↕ 0 P 0 ↕	11001011 00100 r	CB	2	2
shl [.hl] shla	C←[.hl](7) [.hl]←[.hl]*2	↕ ↕ 0 P 0 ↕	11001011 00100110	CB 26	2	5
shl [.ix+d] shla	C←[.ix+d](7) [.ix+d]←[.ix+d]*2	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00100110	DD CB 26	4	7
shl [.iy+d] shla	C←[.iy+d](7) [.iy+d]←[.iy+d]*2	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00100110	FD CB 26	4	7
shr r	C←r(0) r←r/2	↕ ↕ 0 P 0 ↕	11001011 00111 r	CB	2	2
shr [.hl]	C←[.hl](0) [.hl]←[.hl]/2	↕ ↕ 0 P 0 ↕	11001011 00111110	CB 3E	2	5
shr [.ix+d]	C←[.ix+d](0) [.ix+d]←[.ix+d]/2	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00111110	DD CB 3E	4	7
shr [.iy+d]	C←[.iy+d](0) [.iy+d]←[.iy+d]/2	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00111110	FD CB 3E	4	7
shrar	tmp←r(7);C←r(0) r←r/2;r(7)←tmp	↕ ↕ 0 P 0 ↕	11001011 00101 r	CB	2	2
shra [.hl]	tmp←[.hl](7);C←[.hl](0) [.hl]←[.hl]/2;[.hl](7)←tmp	↕ ↕ 0 P 0 ↕	11001011 00101110	CB 2E	2	5
shra [.ix+d]	tmp←[.ix+d](7) C←[.ix+d](0) [.ix+d]←[.ix+d]/2 [.ix+d](7)←tmp	↕ ↕ 0 P 0 ↕	11011101 11001011 ← d → 00101110	DD CB 2E	4	7
shra [.iy+d]	tmp←[.iy+d](7) C←[.iy+d](0) [.iy+d]←[.iy+d]/2 [.iy+d](7)←tmp	↕ ↕ 0 P 0 ↕	11111101 11001011 ← d → 00101110	FD CB 2E	4	7

shl 命令と shla命令はまったく同じものなのでオペランドは同一

A 15　分岐命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
br nn	.pc←nn	• • • • • •	11000011 ← nn_l → ← nn_h →	C3	3	3
bnz nn	if z=0 .pc←nn	• • • • • •	11000010 ← nn_l → ← nn_h →	C2	3	3
bz nn	if z=1 .pc←nn	• • • • • •	11001010 ← nn_l → ← nn_h →	CA	3	3
bnc nn	if c=0 .pc←nn	• • • • • •	11010010 ← nn_l → ← nn_h →	D2	3	3
bc nn	if c=1 .pc←nn	• • • • • •	11011010 ← nn_l → ← nn_h →	DA	3	3
bpo nn	if P/V=0 .pc←nn	• • • • • •	11100010 ← nn_l → ← nn_h →	E2	3	3
bpe nn	if P/V=1 .pc←nn	• • • • • •	11101010 ← nn_l → ← nn_h →	EA	3	3
bp nn	if s=0 .pc←nn	• • • • • •	11110010 ← nn_l → ← nn_h →	F2	3	3
bm nn	if s=1 .pc←nn	• • • • • •	11111010 ← nn_l → ← nn_h →	FA	3	3
br [.hl]	.pc←.hl	• • • • • •	11101001	E9	1	1
br [.ix]	.pc←.ix	• • • • • •	11011101 11101001	DD E9	2	2
br [.iy]	.pc←.iy	• • • • • •	11111101 11101001	FD E9	2	2

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
short br e	.pc←.pc+e	• • • • • •	00011000 ← e−2 →	18	2	3
short bnz e	if z=0 .pc←.pc+e	• • • • • •	00100000 ← e−2 →	20	2	2 3
short bz e	if z=1 .pc←.pc+e	• • • • • •	00101000 ← e−2 →	28	2	2 3
short bnc e	if c=0 .pc←.pc+e	• • • • • •	00110000 ← e−2 →	30	2	2 3
short bc e	if c=1 .pc←.pc+e	• • • • • •	00111000 ← e−2 →	38	2	2 3
dbnz e	.b←.b−1;if .b≠0 .pc←.pc+e	• • • • • •	00010000 ← e−2 →	10	2	2

A.16 コール命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
call nn	[.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11001101 ← nn$_l$ → ← nn$_h$ →	CD	3	5
call nz,nn	if z=0 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11000100 ← nn$_l$ → ← nn$_h$ →	C4	3	3 5
call z,nn	if z=1 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11001100 ← nn$_l$ → ← nn$_h$ →	CC	3	3 5
call nc,nn	if c=0 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11010100 ← nn$_l$ → ← nn$_h$ →	D4	3	3 5
call c,nn	if c=1 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11011100 ← nn$_l$ → ← nn$_h$ →	DC	3	3 5
call po,nn	if P/V=0 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11100100 ← nn$_l$ → ← nn$_h$ →	E4	3	3 5
call pe,nn	if P/V=1 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11101100 ← nn$_l$ → ← nn$_h$ →	EC	3	3 5
call p,nn	if s=0 [sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11110100 ← nn$_l$ → ← nn$_h$ →	F4	3	3 5
call m,nn	if s=1 [.sp−2]←.pc$_l$;[.sp−1]←.pc$_h$ .sp←.sp−2;.pc←nn	• • • • • •	11111100 ← nn$_l$ → ← nn$_h$ →	FC	3	3 5

ニーモニック	命令動作	flags	オペコード		B	C
		S Z H P/V N C	76543210	Hex		
ret	$.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11001001	C9	1	3
ret nz	if z=0 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11000000	C0	1	1 3
ret z	if z=1 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11001000	C8	1	1 3
ret nc	if c=0 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11010000	D0	1	1 3
ret c	if c=1 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11011000	D8	1	1 3
ret po	if P/V=0 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11100000	E0	1	1 3
ret pe	if P/V=1 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11101000	E8	1	1 3
ret p	if s=0 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11110000	F0	1	1 3
ret m	if s=1 $.pc_l \leftarrow [.sp];.pc_h \leftarrow [.sp+1];.sp \leftarrow .sp+2$	• • • • • •	11111000	F8	1	1 3
reti	interrupt return	• • • • • •	11101101 01001101	ED 4D	2	5
retn	Non Maskable Interrupt return	• • • • • •	11101101 01000101	ED 45	2	5
brk k	$[.sp-2] \leftarrow .pc_l;[.sp-1] \leftarrow .pc_h$ $.sp \leftarrow .sp-2;.pc_l \leftarrow k;.pc_h \leftarrow 0$	• • • • • •	11k/s111		1	4

A 17 入出力命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
in .a,[n]	.a←[n]	• • • • • •	11011011 ← n →	DB	2	3
in r,[.c]	r←[.c]	↕ ↕ 0 P 0 •	11101101 01 r 000	ED	2	3
in .f,[.c]	[.c]	↕ ↕ 0 P 0 •	11101101 01110000	ED 70	2	3
in [.hl++],[.c]	[.hl]←[.c];.b←.b−1 .hl←.hl+1	? ↕ ? ? 1 • *1	11101101 10100010	ED A2	2	4
in [.hl−−],[.c]	[.hl]←[.c];.b←.b−1 .hl←.hl−1	? ↕ ? ? 1 • *1	11101101 10101010	ED AA	2	4
inm [.hl++],[.c]	repeat;[.hl]←[.c];.b←.b−1 .hl←.hl+1;until .b=0	? 1 ? ? 1 •	11101101 10110010	ED B2	2	4 3
inm [.hl−−],[.c]	repeat;[.hl]←[.c];.b←.b−1 .hl←.bl−1;until .b=0	? 1 ? ? 1 •	11101101 10111010	ED BA	2	4 3
out [n],.a	[n]←.a	• • • • • •	11010011 ← n →	D3	2	3
out [.c],r	[.c]←r	• • • • • •	11101101 01 r 001	ED	2	3
out [.c],[.hl++]	[.c]←[.hl];.b←.b−1 .hl←.hl+1	? ↕ ? ? 1 • *1	11101101 10100011	ED A3	2	4
out [.c],[.hl−−]	[.c]←[.hl];.b←.b−1 .hl←.hl−1	? ↕ ? ? 1 • *1	11101101 10101011	ED AB	2	4
outm [.c],[.hl++]	repeat;[.c]←[.hl];.b←.b−1 .hl←.hl+1;until .b=0	? 1 ? ? 1 •	11101101 10110011	ED B3	2	4 3
outm [.c],[.hl−−]	repeat;[.c]←[.hl];.b←.b−1 .hl←.hl−1;until .b=0	? 1 ? ? 1 •	11101101 10111011	ED BB	2	4 3

*1.b−1=0 のとき 1、他は 0

in .f,[.c] は.c レジスターが示すポートの内容によってフラグを変えるだけで、その内容はどこにも格納されない

A.18 CPU 制御命令

ニーモニック	命令動作	flags S Z H P/V N C	オペコード 76543210	Hex	B	C
adj .a	adjust to decimal	↕ ↕ ↕ P • ↕	00100111	27	1	1
not .a	.a←NOT .a	• • 1 • 1 •	00101111	2F	1	1
neg .a	.a←NOT .a+1	↕ ↕ ↕ V 1 ↕	11101101 01000100	ED 44	2	2
notc	c←NOT c	• • ? • 0 ↕	00111111	3F	1	1
setc	c←1	• • 0 • 0 1	00110111	37	1	1
nop	NO operation	• • • • • •	00000000	00	1	1
halt	HALT	• • • • • •	01110110	76	1	2
di	IFF←0	• • • • • •	11110011	F3	1	2
ei	IFF←1	• • • • • •	11111011	FB	1	1
im 0	interrupt mode 0	• • • • • •	11101101 01000110	ED 46	2	3
im 1	interrupt mode 1	• • • • • •	11101101 01010110	ED 56	2	3
im 2	interrupt mode 2	• • • • • •	11101101 01011110	ED 5E	2	3

B
R800かけ算命令用マクロ

B.1　R800 のかけ算命令

R800 には、以下のかけ算命令があります。

B.1.1　8 ビットのかけ算

8 ビットレジスタ同士を乗算して、その結果を HL レジスタに返します。かけ算は符号なしで実行されます。インストラクションなどは以下のとおりです。

演算	インストラクション	クロック数
HL←A*B	ED C1	14
HL←A*C	ED C9	14
HL←A*D	ED D1	14
HL←A*E	ED D9	14

B.1.2　16 ビットのかけ算

16 ビットレジスタ同士を乗算して、その結果を DE:HL レジスタに返します。かけ算は符号なしで実行されます。インストラクションなどは以下のとおりです。

演算	インストラクション	クロック数
DE:HL←HL*BC	ED C3	36
DE.HL←HL*SP	ED F3	36

B.2 M80 のかけ算用マクロ

M80 でこれらのかけ算命令を使うときは、以下のようなマクロを定義します。

```
mult8   macro   reg
        ifidn <reg>, <b>
        defb    0edh, 0c1h
        else
        ifidn <reg>, <B>
        defb    0edh, 0c1h
        else
        ifidn <reg>, <c>
        defb    0edh, 0c9h
        else
        ifidn <reg>, <C>
        defb    0edh, 0c9h
        else
        ifidn <reg>, <d>
        defb    0edh, 0d1h
        else
        ifidn <reg>, <D>
        defb    0edh, 0d1h
        else
        ifidn <reg>, <e>
        defb    0edh, 0d9h
        else
        ifidn <reg>, <E>
        defb    0edh, 0d9h
        else
        if1
        .printx *MULT8: illegal argument*
        defb    00h, 00h
        err
        endif
        endif
        endif
        endif
        endif
        endif
        endif
        endif
        endif
        endm

mult16  macro   reg
        ifidn <reg>, <bc>
        defb    0edh, 0c3h
        else
        ifidn <reg>, <BC>
        defb    0edh, 0c3h
        else
        ifidn <reg>, <sp>
        defb    0edh, 0f3h
        else
```

```
ifidn <reg>, <SP>
defb    0edh, 0f3h
else
if1
 printx *MULT16: illegal argument*
defb    00h, 00h
err
endif
endif
endif
endif
endif
endm
```

C

MSXViewファンクション一覧

C.1　ファンクション名順一覧

機能番号	名前	意味	ページ
243	_absread()	論理セクタを用いた読み出し	496
107	_actioncntl()	コントロール実行中の表示	574
183	_arc()	円弧の描画	431
80	_backwin()	ウィンドウを最後面に移動	394
365	_basename()	パス名の解析（ファイル名の獲得）	511
367	_bdos()	MSX-DOS2 システムコールの実行（C 言語対応）	513
130	_blcpixel()	ブロックへの点の書き込み	415
129	_blcpoint()	ブロック上のカラーコードの獲得	415
131	_blcread()	ブロックからメモリへの読み込み	416
132	_blcwrite()	メモリからブロックへの書き込み	416
178	_box()	中を塗りつぶした四角形の描画	429
198	_changecolor()	エリア内の指定色の変更	438
357	_chdir()	カレントディレクトリの変更	504
229	_chfile()	カレントファイルの設定	489
60	_chfont()	カレントフォントの変更	402
371	_chkversion()	MSXView のバージョン番号の検査	515
245	_choice()	ディスクフォーマットメッセージの獲得	496
312	_chpd()	プリンタドライバの変更	624
50	_chpen()	カレントペンの変更	402
140	_chrwidth()	文字幅の獲得	454
269	_chtext()	カレントテキストの変更	470
84	_chwin()	カレントウィンドウの変更	396
36	_chwinker()	カレントウィンカの変更	549

機能番号	名前	意味	ページ
68	_clearrootbd()	ルートボードのクリア	399
76	_clearwin()	ウィンドウのクリア	392
108	_closecntl()	コントロールのクローズ	575
259	_closedlg()	ダイアログボックスのクローズ	595
294	_closeda()	DA メニューのクローズ	603
154	_closemenu()	メニューのクローズ	588
390	_closevb()	VRAM バッファのクローズ	616
74	_closewin()	ウィンドウのクローズ	391
411	_cmkfpath()	ファイル作成用フルパス名の獲得	524
187	_colicon()	指定色によるアイコンの描画	433
192	_copy()	エリアのコピー	435
58	_createfont()	フォントの作成	401
48	_createpen()	ペンの作成	400
265	_createtext()	テキストの作成	467
72	_createwin()	ウィンドウの作成	390
34	_createwinker()	ウィンカの作成	548
247	_currentfile()	カレントファイルの獲得	497
61	_currentfont()	カレントフォントの獲得	403
51	_currentpen()	カレントペンの獲得	402
268	_currenttext()	カレントテキストの獲得	469
85	_currentwin()	カレントウィンドウの獲得	396
39	_currentwinker()	カレントウィンカの獲得	550
197	_curtain()	エリアの塗りつぶし (OR)	437
59	_deletefont()	フォントの削除	401
49	_deletepen()	ペンの削除	401
267	_deletetext()	テキストの削除	469
75	_deletewin()	ウィンドウの削除	392
35	_deletewinker()	ウィンカの削除	548
137	_dfont()	1文字表示	453
186	_dicon()	アイコンの描画	432
168	_direct()	描画エリアの設定	425
364	_dirname()	パス名の解析 (ディレクトリパス名の獲得)	510
105	_dispallcntl()	配列内のコントロールすべての表示	573
104	_dispcntl()	コントロールの表示	573
266	_disptext()	初期化をともなうテキストの表示	468
138	_dkanji()	全角1文字表示	453
260	_dlgselect()	ダイアログ上のアクションの獲得	596
410	_dosexec()	DOS コマンドの実行	606
139	_dpattern()	パターンの表示	453

機能番号	名前	意味	ページ
112	_drivecntl()	コントロールドライバの直接呼び出し	577
295	_drivesystem()	DAの処理	603
141	_dstr()	文字列の表示	454
171	_endtest()	テストモードの終了	427
301	_endview()	MSXView の終了	604
93	_endzoom()	ズームの終了	398
196	_erase()	指定色でのエリアの塗りつぶし	437
195	_erasearea()	カレントペンによるエリアの塗りつぶし	437
373	_errmessage()	メッセージダイアログボックスの表示2	598
251	_execute()	オーバーレイモジュールの実行	499
393	_execute2()	オーバーレイモジュールの実行（任意のファイル）	523
253	_exitmodule()	オーバーレイモジュール・チャイルドプログラムの強制終了	500
241	_falloc()	ファイルバッファの獲得	495
225	fclose()	ファイルのクローズ	487
226	_fcreate()	ファイルの新規作成	488
366	_fcreate2()	ファイルの新規作成（アトリビュート指定あり）	512
227	fdelete()	ファイルの削除	488
238	ferror()	エラーコードの獲得	493
362	_ffirst()	最初のエントリの検索	508
387	_fflush()	ディスクバッファのフラッシュ	522
242	ffree()	ファイルバッファの解放	495
376	fgetattr()	ファイルのアトリビュートの獲得	516
378	_fgetftime()	ファイルの日付と時刻の獲得	517
380	_fhdelete()	ファイルハンドルの削除	518
383	_fhgetattr()	ファイルハンドルのアトリビュートの獲得	520
385	_fhgetftime()	ファイルハンドルの日付と時刻の獲得	521
382	_fhmove()	ファイルハンドルの移動	519
381	_fhrename()	ファイルハンドルの名前の変更	519
384	_fhsetattr()	ファイルハンドルのアトリビュートの設定	520
386	_fhsettime()	ファイルハンドルの日付と時刻の設定	521
182	_filloval()	中を塗りつぶした円の描画	430
185	_fillpai()	中を塗りつぶした扇形の描画	432
190	_fillpolygon()	中を塗りつぶした多角形の描画	434
180	_fillround()	中を塗りつぶした角の丸い四角形の描画	430
111	findcntl()	指定した座標を含むコントロールの検索	576
103	_findpart()	パートナンバーの獲得	572

機能番号	名前	意味	ページ
221	_getdiskinfo()	ディスク情報の獲得	484
219	_getdrive()	デフォルトドライブの獲得	483
12	getevent()	イベント情報の獲得	541
220	_getfileinfo()	ファイル情報の獲得	483
135	_getfontpat()	フォントパターンの獲得	452
289	getjkeyfunc()	機能コードの獲得	610
146	getjispat()	JIS コードによるフォントパターンの読み込み	455
286	_getkeyfunc()	機能コードの獲得	607
19	_getkeyinfo()	キーボード状態の獲得	543
292	_getkeymap()	キーマップの獲得	611
360	_getlogin()	ディスクの接続状況の獲得	505
5	_getpalette()	パレットの獲得	440
27	_getpatnumber()	カレントカーソルの獲得	545
173	[illegible]b()	ラバーバンドカラーの獲得	427
388	_getscreenmode()	スクリーンモードの獲得	407
310	_gettime()	時刻の獲得	605
77	_getwininfo()	ウィンドウ情報の獲得	392
88	_gtol()	グローバル座標からローカル座標への変換	397
44	_hide()	カーソルの消去	551
157	_hilite()	メニュー項目の強調	589
188	_index()	上部のみ角の丸い四角形の描画	433
118	_initblc()	ビットブロックマネージャの初期化	412
98	_initcntl()	コントロールマネージャの初期化	571
21	_initcursor()	カーソル表示の初期化	544
257	_initdlg()	ダイアログマネージャの初期化	595
10	_initevent()	イベントマネージャの初期化	541
240	_initfalloc()	ファイルアロケータの初期化	494
147	_initffile()	フォントファイルアクセスの初期化	455
133	_initfont()	フォントパックの初期化	451
57	_initfonthandle()	フォントハンドルの初期化	400
166	_initgraf()	グラフパックの初期化	425
285	_initkeymap()	キーマップマネージャの初期化	607
303	_initmemory()	メモリマネージャの初期化	614
150	_initmenu()	メニューマネージャの初期化	587
248	_initoverlay()	オーバーレイマネージャの初期化	498
47	_initpenhd()	ペンハンドルの初期化	399
311	_initprint()	プリントマネージャの初期化	624

機能番号	名前	意味	ページ
217	_initres()	リソースマネージャの初期化	482
264	_inittexthd()	テキストハンドルの初期化	467
71	_initwin()	ウィンドウマネージャの初期化	389
33	_initwinker()	ウィンカの初期化	547
279	_inserttext()	文字列の挿入	474
158	_ismenu()	メニュー内のイベントのテスト	590
32	_jobcursor()	ジョブカーソルの設定	547
252	_jump()	MSXView アプリケーションの起動	500
156	_keymenu()	ショートカットキーの処理	589
160	_keypopup()	ポップアップメニューでのショートカットキーの処理	591
29	_killareacursor()	カーソルの有効エリアの削除	546
26	_killpatcursor()	カーソルパターンの削除	545
136	_knjwidth()	全角文字の幅の獲得	452
176	_line()	線の描画	428
275	_locatetext()	テキストカーソルの移動	473
89	_ltog()	ローカル座標からグローバル座標への変換	397
306	_malloc()	メモリブロックの獲得	615
287	_mapcntlkey()	機能コードのマッピング	608
263	_message()	メッセージダイアログボックスの表示	597
359	_mkdir()	ディレクトリの作成	505
361	_mkfpath()	フルパス名の獲得	506
261	_modaldlg()	モーダルダイアログのアクションの獲得	596
256	_modulevalue()	オーバーレイモジュール・チャイルドプログラムからの戻り値の獲得	502
193	_move()	エリアの移動	435
297	_moveframe()	エリアの移動	436
174	_movepen()	ペンの移動	428
90	_movepopup()	ポップアップウィンドウ位置の設定	399
81	_movewin()	ウィンドウの位置の移動	394
239	_msxdos()	MSX-DOS2 システムコールの実行	494
121	_newblc()	新規ブロックの獲得	413
119	_newlot()	新規ロットの割り付け	412
109	_openallcntl()	配列内のコントロールすべてのオープン	575
102	_opencntl()	コントロールのオープン	572
258	_opendlg()	ダイアログボックスの表示	595
293	_openda()	DA メニューのオープン	603
153	_openmenu()	メニューのオープン	587

機能番号	[illegible]	意味	ページ
389	_openvb()	VRAM バッファのオープン	615
73	_openwin()	ウィンドウのオープン	391
181	_oval()	円の描画	430
184	_pai()	扇形の描画	431
368	_pcmplay()	PCMの再生	612
369	_pcmrec()	PCMの録音	612
314	_pd()	プリンタドライバの起動	625
54	_penadrs()	ペン情報の獲得	404
201	_pixel()	カラーコードの獲得	439
189	_polygon()	多角形の描画	433
231	_popfile()	カレントファイルの復帰	490
63	_popfont()	フォントの復帰	404
53	_poppen()	カレントペンの復帰	403
271	_poptext()	テキストの復帰	471
262	_popupdlg()	ポップアップダイアログの表示とアクションの獲得	597
87	_popwin()	カレントウィンドウの復帰	397
38	_popwinker()	カレントウィンカの復帰	549
313	_printinfo()	プリント情報の獲得	624
175	_pset()	点の描画	428
230	_pushfile()	保存をともなうカレントファイルの変更	490
62	_pushfont()	保存をともなうカレントフォントの変更	404
52	_pushpen()	保存をともなうカレントペンの変更	403
270	_pushtext()	保存をともなったカレントテキストの変更	470
86	_pushwin()	保存をともなうカレントウィンドウの変更	396
37	_pushwinker()	保存をともなうカレントウィンカの変更	549
123	_putblc()	ブロックへの保存	417
13	_putevent()	イベント情報のイベントキューへの追加	541
6	_rd_sysdata()	システムデータの読み出し	602
202	_readbit()	エリア内のカラーコードの読み出し	439
148	_readgaiji()	外字ファイルの読み込み	455
392	_readvb()	VRAM バッファからの読み出し	617
278	_redisptext()	テキストの再表示	468
65	_renewfont()	フォントの更新	405
55	_renewpen()	ペンの更新	405
126	_resizeblc()	ブロックサイズの変更	418
82	_resizewin()	ウィンドウサイズの変更	395
128	_restoreblc()	ブロックからの画像の取り出し	414

機能番号	名前	意味	ページ
199	_reverse()	エリアの塗りつぶし (XOR)	438
179	_round()	角の丸い四角形の描画	429
200	_roundarea()	エリアにしたがった角の丸い四角形の描画	439
304	_sbrk()	メモリブロックの獲得	614
2	_screen()	スクリーンモードの設定	406
374	_screensize()	スクリーンサイズの獲得	407
191	_scroll()	エリア内のスクロール	434
277	_scrolltext()	スクロール処理	474
155	_selectmenu()	メニュー項目が選択されたときの処理	588
159	_selectpopup()	ポップアップメニューの処理	590
28	_setareacursor()	カーソルの有効領域の変更	546
101	_setcntl()	カスタムコントロールの割り付け	571
276	_setcursor()	バッファ中のテキストカーソルの移動	473
307	_setdate()	日付の設定	604
20	_setdevice()	ポインティングデバイスの設定	544
218	_setdrive()	デフォルトドライブの設定	482
134	_setfont()	フォントスタイルの変更	451
18	_setkeyinfo()	キーボード状態の設定	543
291	_setkeymap()	キーマップの設定	610
4	_setpalette()	パレットの設定	440
25	_setpatcursor()	カーソルパターンの変更	545
167	_setpen()	ペンの設定	425
172	_setrub()	ラバーバンドモードの開始・終了	427
280	_settextcursor()	テキスト位置の設定	475
309	_settime()	時刻の設定	605
78	_setwininfo()	ウィンドウの変更	393
43	_show()	カーソルの表示	550
372	_showtitle()	タイトルの表示	605
416	_signal()	物理エラー処理ルーチンの設定	526
14	_snsevent()	イベント発生の調査	542
127	_storeblc()	ブロックへの画像の保存	414
142	_strwidth()	文字列の幅の獲得	454
125	_swapblc()	ブロック内と画面の画像の交換	418
45	_sync()	次の垂直同期までの待機	551
254	_system()	チャイルドプログラムの起動	501
31	_systemcursor()	システムカーソルの設定	547
170	_testarea()	エリアの重なりをテストするモードの開始	426

機能番号	[illegible]	[illegible]	ページ
110	_testcntl()	任意の座標がコントロールに含まれるかの検索	576
169	_testpos()	図形の重なりをテストするモードの開始	426
272	_textadrs()	テキスト構造体の獲得	471
274	_texteditfunc()	編集ファンクションの実行	472
106	_trackcntl()	コントロールの実行	574
15	_ungetevent()	イベント情報のイベントキューへの返還	542
40	_winkeradrs()	ウィンカ情報の獲得	550
1	_wr_sysdata()	システムデータの書き込み	602
203	_writebit()	エリアへのカラーコードの書き込み	440
273	_writetext()	テキストの編集	472
391	_writevb()	VRAM バッファへの書き込み	616
91	_zoom()	エリアのズーム	398
92	_zoomwin()	ウィンドウのズーム	398

C.2 機能番号順一覧

機能番号	名前	意味	ページ
1	_wr_sysdata()	システムデータの書き込み	602
2	_screen()	スクリーンモードの設定	406
4	_setpalette()	パレットの設定	440
5	_getpalette()	パレットの獲得	440
6	_rd_sysdata()	システムデータの読み出し	602
10	_initevent()	イベントマネージャの初期化	541
12	_getevent()	イベント情報の獲得	541
13	_putevent()	イベント情報のイベントキューへの追加	541
14	_susevent()	イベント発生の調査	542
15	_ungetevent()	イベント情報のイベントキューへの返還	542
16	_getcoord()	カーソル座標の獲得	542
17	_flushevents()	イベントキューのクリア	543
18	_setkeyinfo()	キーボード状態の設定	543
19	_getkeyinfo()	キーボード状態の獲得	543
20	_setdevice()	ポインティングデバイスの設定	544
21	_initcursor()	カーソル表示の初期化	544
22	_getdevice()	ポインティングデバイスの獲得	544
25	_setpatcursor()	カーソルパターンの変更	545
26	_killpatcursor()	カーソルパターンの削除	545
27	_getpath()	カレントパスの獲得	545
	_setareacursor()	カーソルの有効領域の変更	546
29	_killareacursor()	カーソルの有効エリアの削除	546
30	_getareanumber()	カーソルがあるエリアの獲得	546
31	_systemcursor()	システムカーソルの設定	547
32	_jobcursor()	ジョブカーソルの設定	547
33	_initwinker()	ウィンカの初期化	547
34	_createwinker()	ウィンカの作成	548
35	_deletewinker()	ウィンカの削除	548
36	_chwinker()	カレントウィンカの変更	549
37	_pushwinker()	保存をともなうカレントウィンカの変更	549
38	_popwinker()	カレントウィンカの復帰	549
39	_currentwinker()	カレントウィンカの獲得	550
40	_winkeradrs()	ウィンカ情報の獲得	550
43	_show()	カーソルの表示	550
44	_hide()	カーソルの消去	551
45	_sync()	次の垂直同期までの待機	551

機能番号	名前	意味	ページ
47	_initpenhd()	ペンハンドルの初期化	399
48	_createpen()	ペンの作成	400
49	_deletepen()	ペンの削除	401
50	_chpen()	カレントペンの変更	402
51	_currentpen()	カレントペンの獲得	402
52	_pushpen()	保存をともなうカレントペンの変更	403
53	_poppen()	カレントペンの復帰	403
54	_penadrs()	ペン情報の獲得	404
55	_renewpen()	ペンの更新	405
56	_getdefpen()	デフォルトペンの獲得	406
57	_initfonthandle()	フォントハンドルの初期化	400
58	_createfont()	フォントの作成	401
59	_deletefont()	フォントの削除	401
60	_chfont()	カレントフォントの変更	402
61	_currentfont()	カレントフォントの獲得	403
62	_pushfont()	保存をともなうカレントフォントの変更	404
63	_popfont()	フォントの復帰	404
64	_fontadrs()	フォント情報の獲得	405
65	_renewfont()	フォントの更新	405
66	_getdeffont()	デフォルトフォントの獲得	406
68	_clearrootbd()	ルートボードのクリア	399
71	_initwin()	ウィンドウマネージャの初期化	389
72	_createwin()	ウィンドウの作成	390
73	_openwin()	ウィンドウのオープン	391
74	_closewin()	ウィンドウのクローズ	391
75	_deletewin()	ウィンドウの削除	392
76	_clearwin()	ウィンドウのクリア	392
77	_getwininfo()	ウィンドウ情報の獲得	392
78	_setwininfo()	ウィンドウの変更	393
79	_frontwin()	ウィンドウを最前面に移動	393
80	_backwin()	ウィンドウを最後面に移動	394
81	_movewin()	ウィンドウの位置の移動	394
82	_resizewin()	ウィンドウサイズの変更	395
83	_findwin()	ウィンドウハンドルの獲得	395
84	_chwin()	カレントウィンドウの変更	396
85	_currentwin()	カレントウィンドウの獲得	396
86	_pushwin()	保存をともなうカレントウィンドウの変更	396
87	_popwin()	カレントウィンドウの復帰	397

機能番号	名前	意味	ページ
88	_gtol()	グローバル座標からローカル座標への変換	397
89	_ltog()	ローカル座標からグローバル座標への変換	397
90	_movepopup()	ポップアップウィンドウ位置の設定	399
91	_zoom()	エリアのズーム	398
92	_zoomwin()	ウィンドウのズーム	398
93	_endzoom()	ズームの終了	398
98	_initcntl()	コントロールマネージャの初期化	571
101	_setcntl()	カスタムコントロールの割り付け	571
102	_opencntl()	コントロールのオープン	572
103	findpart()	パートナンバーの獲得	572
104	_dispcntl()	コントロールの表示	573
105	_dispallcntl()	配列内のコントロールすべての表示	573
106	trackcntl()	コントロールの実行	574
107	_actioncntl()	コントロール実行中の表示	574
108	_closecntl()	コントロールのクローズ	575
109	_openallcntl()	配列内のコントロールすべてのオープン	575
110	testcntl()	任意の座標がコントロールに含まれるかの検索	576
111	_findcntl()	指定した座標を含むコントロールの検索	576
112	_drivecntl()	コントロールドライバの直接呼び出し	577
118	_initblc()	ビットブロックマネージャの初期化	412
119	_newlot()	新規ロットの割り付け	412
120	_freelot()	ロットの解放	413
121	_newblc()	新規ブロックの獲得	413
122	_freeblc()	ブロックの解放	413
123	_putblc()	ブロックへの保存	417
124	_getblc()	ブロックから画面への表示	417
125	_swapblc()	ブロック内と画面の画像の交換	418
126	_resizeblc()	ブロックサイズの変更	418
127	_storeblc()	ブロックへの画像の保存	414
128	_restoreblc()	ブロックからの画像の取り出し	414
129	_blcpoint()	ブロック上のカラーコードの獲得	415
130	_blcpixel()	ブロックへの点の書き込み	415
131	_blcread()	ブロックからメモリへの読み込み	416
132	_blcwrite()	メモリからブロックへの書き込み	416
133	_initfont()	フォントパックの初期化	451
134	_setfont()	フォントスタイルの変更	451
135	_getfontpat()	フォントパターンの獲得	452

機能番号	名前	意味	ページ
136	_knjwidth()	全角文字の幅の獲得	452
137	_dfont()	1 文字表示	453
138	_dkanji()	全角 1 文字表示	453
139	_dpattern()	パターンの表示	453
140	_chrwidth()	文字幅の獲得	454
141	_dstr()	文字列の表示	454
142	_strwidth()	文字列の幅の獲得	454
146	_getjispat()	JIS コードによるフォントパターンの読み込み	455
147	_initfile()	フォントファイルアクセスの初期化	455
148	readgaiji()	外字ファイルの読み込み	455
150	_initmenu()	メニューマネージャの初期化	587
153	[illegible]menu()	メニューのオープン	587
154	_closemenu()	メニューのクローズ	588
155	_selectmenu()	メニュー項目が選択されたときの処理	588
156	keymenu()	ショートカットキーの処理	589
157	_hilite()	メニュー項目の強調	589
158	_ismenu()	メニュー内のイベントのテスト	590
159	_selectpopup()	ポップアップメニューの処理	590
160	keypopup()	ポップアップメニューでのショートカットキーの処理	591
166	_initgraf()	グラフパックの初期化	425
167	_setpen()	ペンの設定	425
168	_direct()	描画エリアの設定	425
169	testpos()	図形の重なりをテストするモードの開始	426
170	testarea()	エリアの重なりをテストするモードの開始	426
171	_endtest()	テストモードの終了	427
172	_setrub()	ラバーバンドモードの開始・終了	427
173	_getrub()	ラバーバンドカラーの獲得	427
174	_movepen()	ペンの移動	428
175	pset()	点の描画	428
176	line()	線の描画	428
177	_frame()	四角形の描画	429
178	box()	中を塗りつぶした四角形の描画	429
179	_round()	角の丸い四角形の描画	429
180	_fillround()	中を塗りつぶした角の丸い四角形の描画	430
181	_oval()	円の描画	430
182	_filloval()	中を塗りつぶした円の描画	430

機能番号	名前	意味	ページ
183	_arc()	円弧の描画	431
184	_pai()	扇形の描画	431
185	_fillpai()	中を塗りつぶした扇形の描画	432
186	_dicon()	アイコンの描画	432
187	_colicon()	指定色によるアイコンの描画	433
188	index()	上部のみ角の丸い四角形の描画	433
189	polygon()	多角形の描画	433
190	fillpolygon()	中を塗りつぶした多角形の描画	434
191	_scroll()	エリア内のスクロール	434
192	_copy()	エリアのコピー	435
193	_move()	エリアの移動	435
194	_framearea()	エリアにしたがった四角形の描画	436
195	_erasearea()	カレントペンによるエリアの塗りつぶし	437
196	_erase()	指定色でのエリアの塗りつぶし	437
197	_curtain()	エリアの塗りつぶし (OR)	437
198	_changecolor()	エリア内の指定色の変更	438
199	_reverse()	エリアの塗りつぶし (XOR)	438
200	_roundarea()	エリアにしたがった角の丸い四角形の描画	439
201	pixel()	カラーコードの獲得	439
202	readbit()	エリア内のカラーコードの読み出し	439
203	writebit()	エリアへのカラーコードの書き込み	440
217	_initres()	リソースマネージャの初期化	482
218	_setdrive()	デフォルトドライブの設定	482
219	_getdrive()	デフォルトドライブの獲得	483
220	_getfileinfo()	ファイル情報の獲得	483
221	_getdiskinfo()	ディスク情報の獲得	484
222	_fset()	ファイルの検索	485
223	fnext()	次のファイルの検索	486
224	_fopen()	ファイルのオープン	487
225	fclose()	ファイルのクローズ	487
226	_fcreate()	ファイルの新規作成	488
227	_fdelete()	ファイルの削除	488
228	frename()	ファイル名・ディレクトリ名の変更	489
229	_chfile()	カレントファイルの設定	489
230	_pushfile()	保存をともなうカレントファイルの変更	490
231	_popfile()	カレントファイルの復帰	490
232	_fwrite()	カレントファイルへの書き込み	491
233	_fread()	カレントファイルからの読み込み	491

機能番号	名前		ページ
235	_fseek()	ファイルポインタの設定	492
236	_fpoint()	ファイルポインタの獲得	492
237	_fsize()	ファイルサイズの獲得	493
238	_ferror()	エラーコードの獲得	493
239	_msxdos()	MSX-DOS2 システムコールの実行	494
240	_initfalloc()	ファイルアロケータの初期化	494
241	_falloc()	ファイルバッファの獲得	495
242	_ffree()	ファイルバッファの解放	495
243	_absread()	論理セクタを用いた読み出し	496
245	_choice()	ディスクフォーマットメッセージの獲得	496
246	_format()	ディスクのフォーマット	497
247	_currentfile()	カレントファイルの獲得	497
248	_initoverlay()	オーバーレイマネージャの初期化	498
251	_execute()	オーバーレイモジュールの実行	499
252	_jump()	MSXView アプリケーションの起動	500
253	_exitmodule()	オーバーレイモジュール・チャイルドプログラムの強制終了	500
254	_system()	チャイルドプログラムの起動	501
256	_modulevalue()	オーバーレイモジュール・チャイルドプログラムからの戻り値の獲得	502
257	_initdlg()	ダイアログマネージャの初期化	595
258	_opendlg()	ダイアログボックスの表示	595
259	_closedlg()	ダイアログボックスのクローズ	595
260	_dlgselect()	ダイアログ上のアクションの獲得	596
261	_modaldlg()	モーダルダイアログのアクションの獲得	596
262	_popupdlg()	ポップアップダイアログの表示とアクションの獲得	597
263	_message()	メッセージダイアログボックスの表示	597
264	_inittexthd()	テキストハンドルの初期化	467
265	_createtext()	テキストの作成	467
266	_disptext()	初期化をともなうテキストの表示	468
267	_deletetext()	テキストの削除	469
268	_currenttext()	カレントテキストの獲得	469
269	_chtext()	カレントテキストの変更	470
270	_pushtext()	保存をともなったカレントテキストの変更	470
271	_poptext()	テキストの復帰	471
272	_textadrs()	テキスト構造体の獲得	471
273	_writetext()	テキストの編集	472

機能番号	名前	意味	ページ
274	_texteditfunc()	編集ファンクションの実行	472
275	_locatetext()	テキストカーソルの移動	473
276	setcursor()	バッファ中のテキストカーソルの移動	473
277	_scrolltext()	スクロール処理	474
278	_redisptext()	テキストの再表示	468
279	_inserttext()	文字列の挿入	474
280	_settextcursor()	テキスト位置の設定	475
284	_flushjsystem()	漢字変換のキャンセル	475
285	_initkeymap()	キーマップマネージャの初期化	607
286	_getkeyfunc()	機能コードの獲得	607
287	_mapcntlkey()	機能コードのマッピング	608
288	_getalphkey()	機能コードがマッピングされているキーの獲得	610
289	getjekyfunc()	機能コードの獲得	610
291	_setkeymap()	キーマップの設定	610
292	_getkeymap()	キーマップの獲得	611
293	_openda()	DA メニューのオープン	603
294	_closeda()	DA メニューのクローズ	603
295	_drivesystem()	DA の処理	603
297	_moveframe()	エリアの移動	436
300	_fmenu()	ファイルの選択	503
301	_endview()	MSXView の終了	604
303	_initmemory()	メモリマネージャの初期化	614
304	_sbrk()	メモリブロックの獲得	614
305	_free()	メモリブロックの解放	615
306	_malloc()	メモリブロックの獲得	615
307	_setdate()	日付の設定	604
308	getdate()	日付の獲得	604
309	_settime()	時刻の設定	605
310	_gettime()	時刻の獲得	605
311	_initprint()	プリントマネージャの初期化	624
312	_chpd()	プリンタドライバの変更	624
313	_printinfo()	プリント情報の獲得	624
314	_pd()	プリンタドライバの起動	625
357	_chdir()	カレントディレクトリの変更	504
358	_getcwd()	カレントワーキングディレクトリの獲得	504
359	_mkdir()	ディレクトリの作成	505
360	getlogin()	ディスクの接続状況の獲得	505

機能番号	名前	意味	ページ
361	_mkfpath()	フルパス名の獲得	506
362	_ffirst()	最初のエントリの検索	508
363	_fnext2()	次のエントリの検索	509
364	_dirname()	パス名の解析（ディレクトリパス名の獲得）	510
365	_basename()	パス名の解析（ファイル名の獲得）	511
366	fcreate2()	ファイルの新規作成（アトリビュート指定あり）	512
367	bdos()	MSX DOS2 システムコールの実行（C 言語対応）	513
368	_pcmplay()	PCMの再生	612
369	_pcmrec()	PCM の録音	612
370	_fnew()	新しいエントリの検索	514
371	_chkversion()	MSXView のバージョン番号の検査	515
372	_showtitle()	タイトルの表示	605
373	_errmessage()	メッセージダイアログボックスの表示2	598
374	_screensize()	スクリーンサイズの獲得	407
375	fmove()	ファイルの移動	515
376	fgetattr()	ファイルのアトリビュートの獲得	516
377	_fsetattr()	ファイルのアトリビュートの設定	516
378	_fgetftime()	ファイルの日付と時刻の獲得	517
379	_fsettime()	ファイルの日付と時刻の設定	518
380	fhdelete()	ファイルハンドルの削除	518
381	_fhrename()	ファイルハンドルの名前の変更	519
382	fhmove()	ファイルハンドルの移動	519
383	fhgetattr()	ファイルハンドルのアトリビュートの獲得	520
384	_fhsetattr()	ファイルハンドルのアトリビュートの設定	520
385	_fhgetftime()	ファイルハンドルの日付と時刻の獲得	521
386	_fhsettime()	ファイルハンドルの日付と時刻の設定	521
387	_fflush()	ディスクバッファのフラッシュ	522
388	_getscreenmode()	スクリーンモードの獲得	407
389	_openvb()	VRAM バッファのオープン	615
390	_closevb()	VRAM バッファのクローズ	616
391	writevb()	VRAM バッファへの書き込み	616
392	readvb()	VRAM バッファからの読み出し	617
393	_execute2()	オーバーレイモジュールの実行（任意のファイル）	523
410	_dosexec()	DOS コマンドの実行	606
411	_cmkfpath()	ファイル作成用フルパス名の獲得	524

機能番号	名前	意味	ページ
414	_fpathset()	複数パスからのファイルの検索の設定	525
415	_fpathnext()	複数パスからのファイルの検索	526
416	_signal()	物理エラー処理ルーチンの設定	526

注意　機能番号の空いている部分はシステム予約のファンクションです。システム予約のファンクションをコールした場合の結果については保証されません。

D
サンプルプログラム

サンプルプログラムに入っているプログラム（以下、サンプルプログラム）は、「MSX-Datapack turbo R 版」の登録ユーザーの方であれば、ユーザープログラムに組み込んで使用することができます。そのユーザープログラムを販売・頒布する場合も、弊社との契約は必要ありません。また、その際、弊社の Copyright 表示なども必要ありません。

ただし、サンプルプログラムの著作権は株式会社アスキーが保有します。したがって、一部または全部に関わらず、サンプルプログラムの内容をそのまま、単体で第三者に販売・頒布することは禁止します。同様に、パソコン通信において、サンプルプログラムのソースコードまたはオブジェクトプログラムを単体でホストコンピュータにアップロードすることは禁止します。

サンプルプログラムの内容およびマニュアルの正誤情報などは、サンプルディスクのファイルに記録します。サンプルプログラムを使う前に、以下のファイルをご覧下さい。

表 4.12 サンプルプログラムの情報を記録したファイル

ファイル名	内容
README.DOC	マニュアルの正誤情報など
CONTENTS.DOC	サンプルディスクに含まれるファイルの内容

これらは、漢字を含んだファイルです。内容を参照するときは、KID.COM などのエディタやそれに類するユーティリティを使用するか、MSX-DOS または BASIC を漢字モードにして、ファイルを表示して下さい。

また、サンプルプログラムを運用した結果の影響については、弊社は責任を負いませんので、ご了承下さい。

索引

■サ

お問い合わせについて

弊社では厳重に梱包した上、細心の注意を払って製品を発送しております。万一、輸送上のトラブルが起こった場合にはご一報いただければ新しいものと交換いたします。

マニュアル作成にあたり、なるべく詳細な説明をするように心がけたつもりですが、理解できないところは、実際にコンピュータと向き合って納得のゆくまで確めて下さい。また、他のページを参照するのも1つの方法です。それでも疑問点が解決できないときは、封書にて、下記の要領でお問い合わせ下さい。このパッケージの性格上、電話でのお答は不可能と存しますので、恐れ入りますが、ご了承下さいますようお願い申し上げます。

また、本製品以外に対してのご意見、ご希望がありましたら、弊社までお寄せ下さい。

【記】

1. 送付先

〒107 24　東京都港区南青山6-11-1　スリーエフ南青山ビル
株式会社アスキー ユーザーサポート係

2 必要事項

(a) お客様の氏名、住所（郵便番号）、電話番号（市外局番も含む）

(b) 製品名、製品シリアル番号、ユーザーID番号

(c) 機器構成
本体装置名、メモリバイト数
CRT装置名、フロッピーディスク装置名
プリンタ装置名
その他I/O、I/F装置名

(d) お問い合わせ内容
お問い合わせの内容は、できるだけ製品のマニュアルに記述されている用語を用いて、具体的かつ明確に記述して下さい。なお、障害と思われる現象については、その現象を再現可能な情報が必要です。当社で再現できないものは、調査ができません。その現象が発生するまでの操作手順、データを必ず添付して下さい。データディスクがある場合は、そのコピーも同封していただくと調査がスピーディになります。

また、お客様固有と思われるアプリケーションの設計、作成、運用、保守については、当社のサポート範囲外ですので、お問い合わせいただいても回答できません。例えば、「このプログラムリストはなぜ動かないのか」といったご質問にはお答えできません。ご承知下さいますようお願いいたします。

MSX Datapack Volume 3

1991年12月1日　第1版第1刷
編　集　株式会社アスキー　システム事業部
　　　　担当　松本 有子、北浦 訓行
発行所　株式会社アスキー
　　　　〒107-24　東京都港区南青山6-11-1　スリーエフ南青山ビル

制　作　株式会社ジャパックスインターナショナル